Gedistribueerde patroondetectie en datapublicatie in de biowetenschappen met behulp van bigdatatechnologieën

Distributed Pattern Mining and Data Publication in Life Sciences Using Big Data Technologies

Dieter De Witte

Promotoren: prof. dr. E. Mannens, prof. dr. ir. J. Fostier
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. K. De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017 - 2018

ISBN 978-94-6355-143-4
NUR 984, 983
Wettelijk depot: D/2018/10.500/61

Members of the examination committee

Chair

prof. dr. ir. Filip De Turck (Ghent University)

Secretary

prof. dr. ir. Ruben Verborgh (Ghent University)

Reading Committee

prof. dr. Wesley De Neve (Ghent University)
dr. ir. Laurens De Vocht (Arcelor Mittal)
prof. dr. Yvan Saeys (Ghent University)

Other members

dr. Azarakhsh Jalalvand (Ghent University)
prof dr. Erik Mannens (Ghent University, *promotor*)
prof dr. ir. Jan Fostier (Ghent University, *promotor*)

Preface

I always did something I was a little not ready to do. I think that's how you grow.

—Marissa Mayer

I have added the quote of Marissa Mayer as I think it is a genuine advice to anyone starting their (scientific) career. For some people, including myself, doubting on your own abilities might be the major roadblock in your personal development and might prevent you from pursuing your dreams. The quote is all the stronger as it comes from one of the most successful women in Silicon Valley.[1]

If I have championed one discipline over the years it was that of 'not being ready'. However, I have always been incredibly lucky to have people who convinced me to pursue each of the opportunities which came my way, even when I was at the verge of failing.

My parents deserve special credit here. As they will confirm, there have been numerous occasions where it seemed that I had pushed my luck one step too far. Nonetheless both of them gave me the opportunity to recover from my failures, by granting me a lot of trust and second-chances. I have to also thank my sister for not requiring these second-chances, she must have given me some leeway in doing so.

At university it was Femke Olyslager who managed to inspire and encourage me to become a researcher. This way she had an incredible impact on my professional life. I am also deeply saddened by the fact that she is no longer with us to witness my graduation. Ignace Bogaert provided guidance during my first year as a researcher

[1] Marissa Mayer started her career at Google and played a key role in many of the products we use on a daily basis: Google search, Google maps, Gmail,… At 2012, when she was 27 years old, she became the CEO of Yahoo, making her one of the most influential Women in technology.

and should definitely be credited for introducing me to the Star Trek franchise. I am grateful to Daniel De Zutter who supported my switch to the field of bioinformatics and Piet Demeester and Bart Dhoedt for offering me this great opportunity.

Jan Fostier is responsible for laying the foundations of my career as a researcher. He introduced me to the world of (parallel) algorithms, bioinformatics and scientific writing. Klaas Vandepoele's vast knowledge of the field of computational biology and his enthusiasm about comparative genomics was thought-provoking.

Struggling to get the fruits of my research published, I left to work at Telenet for one year. I met Erik Mannens during one of my trips to Mechelen and he managed to convince me to return to imec and join IDLab. Erik's style of granting a lot of trust while keeping the expectations high, really drives many people to excellence. Especially the creation of the Big Data Science course, I consider a major achievement.

Under the guidance of Ruben Verborgh and Laurens De Vocht I quickly discovered a number of interesting research questions in the field of the Semantic Web Technologies, which finally led to this dissertation. I would also like to thank Hélène for keeping me focused towards this end goal.

My career at Ghent University has been filled with interactions with many interesting people so I would like to thank all colleagues from the Semantic Web office, the people in the Machine Learning office, the people working on Pattern Mining, the people of the Bioinformatics research unit and all people working in a supporting role. Joeri Ruyssinck gets special credit for buying me a hippo (sculpture) when I left the bioinformatics group.

I would also like to thank all jury members for providing me feedback and suggestions to help me improve my manuscript.

Last but not least, I would like to thank my partner Sofie Desmet. She has been a very loving and patient(!) life companion. She gave me 2 beautiful (and stubborn!) children, Astor and Mona, and together with them she's the most important thing in my life.

Dieter De Witte
June 2018,
Gent, Belgium

Contents

List of Acronyms

AP	Anchor Point.
API	Application Programming Interface.
bdi	Brachypodium distachyon.
BGP	Basic Graph Pattern.
BLAST	Basic Local Alignment Search Tool.
BLS	Branch Length Score.
CPU	Central Processing Unit.
DNA	Deoxyribonucleic acid.
ETL	Extract Transform Load.
FDR	False Discovery Rate.
FP	False Positive.
FPR	False Positive Rate.
GHM	Gene Homology Matrix.
GST	Generalized Suffix Tree.
HOX	Homeobox.
HPC	High Performance Computing (infrastructure).
i-ADHoRe	iterative and Automatic Detection of Homologous Regions.
LOD	Linked Open Data.
MPI	Message Passing Interface.
MSA	Multiple Sequence Alignment.
NW	Needleman-Wunsch.
osa	Oryza sativa ssp. indica.
PWM	Position Weight Matrix.
RDF	Resource Description Framework.
RML	RDF Mapping Language.

RNA	Ribonucleic acid.
sbi	Sorghum bicolor.
SPARQL	SPARQL Protocol and RDF Query Language.
SQL	Structured Query Language.
TF	Transcription Factor.
TPF	Triple Pattern Fragments.
URI	Uniform Resource Identifier.
W3C	World Wide Web Consortium.
WatDiv	Waterloo SPARQL Diversity Test Suite.
WGA	Whole-Genome Alignment.
WGD	Whole-Genome Duplication.
zma	Zea mays.

Samenvatting

Het domein van de biowetenschappen bevindt zich in een toestand die veel gelijkenissen vertoont met die van de natuurkunde in de zeventiende eeuw. In een duizelingwekkend tempo volgen de doorbraken elkaar op. Dit wordt nog verder in de hand gewerkt door simultane innovaties in bigdatatechnologieën.

De bigdatahype zorgt voor een renaissance in het landschap van de datamanagementsystemen. Daar ligt de focus nu op een horizontaal schaalbare aanpak. Hierbij beheren meerdere verbonden systemen elk een deel van de data, in plaats van een enkel krachtig systeem.

Het zoeken naar patronen in dergelijke systemen vergt evenwel nieuwe parallelle algoritmes, die optimaal gebruik kunnen maken van deze nieuwe systeemarchitecturen. Dit alles leidt tot een feedbacklus: hoe beter men in staat is om (biologische) data te analyseren en op te slaan, des te meer zal men geneigd zijn om nog meer van die data te gaan genereren. Ook de technologie die nodig is om deze data te bekomen, legt niet langer enige beperking op: de kostprijs voor de sequentiëring van een genoom is in een recordtijd gezakt tot minder dan $1000.

De aandacht voor Big Data hoeft niet te verbazen. Enkele internetgiganten zoals Google en Amazon hebben er hun succes aan te danken. Hun model wordt dan ook gretig overgenomen in allerhande bedrijfstakken, maar ook binnen de wetenschappen. Het is immers minstens even belangrijk om te investeren in het verzamelen van voldoende data, dan om tijd te spenderen aan het ontwikkelen van gesofisticeerde predictie-algoritmes. Hoewel de bigdata-algoritmes vaak iets eenvoudiger zijn, bestaat de uitdaging erin om die algoritmes op een ‘schaalbare’ manier te laten werken. Als we de hoeveelheid data verdubbelen en tegelijk ook de hoeveelheid computationele middelen, streven we er dus naar om de rekentijd in de mate van het mogelijke constant te houden.

In deel I van dit werk worden twee technologieën onderzocht, die toelaten om op een gedistribueerde manier data te gaan verwerken: MPI (Message Passing Interface) en MapReduce. MPI is een protocol, dat een cluster van rekennodes de mogelijkheid biedt om op verschillende manieren boodschappen te versturen. Dat kan volgens puntsgewijze communicatie (partij a zendt een boodschap naar partij b), maar ook via collectieve communicatie waarbij alle partijen simultaan betrokken zijn. MapReduce heeft de interesse in big data in een stroomversnelling gebracht toen duidelijk werd dat het centraal stond binnen de strategie voor dataverwerking bij Google. MapReduce is een veel eenvoudiger paradigma, maar het neemt de eindgebruiker een heleboel verantwoordelijkheden uit handen, dit in tegenstelling tot MPI. Zo zorgt het MapReduce framework voor een gelijkmatige verdeling van het werk en de data over de cluster en voor het opvolgen van de status van de verschillende rekennodes, om zo hardware problemen te verhelpen. Binnen deze doctoraatsthesis bleek de extra flexibiliteit van MPI niet op te wegen tegen de net beschreven voordelen die MapReduce biedt.

In deel II van dit onderzoek worden twee parallelle algoritmes besproken die voor dit proefschrift werden ontwikkeld. Beide algoritmes proberen patronen te ontdekken in DNA sequenties. Het eerste algoritme, genaamd i-ADHoRe, vergelijkt de genomen van verschillende organismen, maar doet dit door te focussen op sequenties van genen. Het algoritme probeert vooral collineaire regio's op te sporen. Dat zijn gensequenties die sterke gelijkenissen vertonen tussen twee chromosomen. Die regio's bieden inzicht in de evolutie van genomen doorheen de tijd en hoe een gemeenschappelijke voorouder van twee organismen er mogelijks uitzag. i-ADHoRe is vooral goed in het detecteren van grootschalige genoomduplicaties. Daarvan is geweten dat zij een belangrijke rol hebben gespeeld in het verhogen van de overlevingskansen van soorten tijdens periodes van massa-extinctie. Het i-ADHoRe algoritme maakt gebruik van MPI om het werk te parallelliseren en is zo in staat om de meest uitdagende datasets binnen het uur te verwerken. Het algoritme heeft ook een hogere sensitiviteit dan andere benaderingen. Dat blijkt o.a. uit de detectie van enkele goedgekende biologische patronen.

Het tweede algoritme zoomt verder in tot op het niveau van de DNA-baseparen. Het BLSSpeller-algoritme gaat op zoek naar korte DNA-sequenties, die we regulatorische sequenties noemen. Die sequenties bevinden zich typisch in de promotor van een gen. Dit is een stukje niet-coderend DNA dat zich net voor het gen bevindt, maar dat evenwel duizenden baseparen kan beslaan. Deze regulatorische sequenties beïn-

vloeden de frequentie waarmee een stuk coderend DNA (= gen) kan vertaald worden naar RNA. Dit proces heet transcriptie. Het beïnvloeden van de frequentie gebeurt vooral bij het initiëren van de transcriptie. De molecule die verantwoordelijk is voor transcriptie, RNA-Polymerase, kan dankzij die regulatorische sequenties vlotter binden op de DNA-sequentie. Er bestaan ook sequenties die die binding verhinderen. Het BLSSpeller-algoritme is een exacte en exhaustieve methode om alle mogelijke patronen op te lijsten. Anders dan zijn voorgangers is het een algoritme dat zich specifiek richt op sequenties van nauwverwantBLe organismen, zoals de eenzaadlobbigen (monocotylen), die het uitgangspunt vormen in deze doctoraatsthesis. Het exacte algoritme werkt door alle mogelijke patronen in het DNA een voor een te testen, weliswaar met enkele slimme branch-and-bound-condities die de zoekruimte sterk beperken en het algoritme aldus versnellen. Een ander verschil met voorgaande algoritmes is dat BLSSpeller nagaat in hoeverre eenzelfde patroon in elk van die genfamilies aanwezig is. Dit laat toe om elk kandidaat-patroon een genoomwijde score toe te kennen. Het BLSSpeller-algoritme werd zowel met MPI als met Map-Reduce geparallelliseerd. Enkel de laatste benadering liet toe om een voldoende grote zoekruimte van patronen te verkennen.

In deel III van dit werk wordt onderzocht hoe de patronen, die gevonden worden met de algoritmes uit deel II, kunnen worden gepubliceerd op een kostenefficiënte en schaalbare manier. In deze context wordt semanticwebtechnologie onderzocht. Die technologie laat toe data te publiceren, op een gedecentraliseerde manier, waarbij integratie en consumptie toch mogelijk blijven. Aangezien de biowetenschappen een erg multidisciplinair gebied zijn, worden die eigenschappen des te belangrijker met het oog op herbruikbaarheid van de resultaten.

Binnen het SEQUEL-project, in samenwerking met Ontoforce, werd onderzocht in hoeverre de huidige technologieën in staat zijn om te werken voor een reeks artificiële en echte biomedische workloads. De ultieme vraag die men daarin hoopt te beantwoorden is in hoeverre een gedecentraliseerde interactie met Linked Data op een interactieve manier kan gebeuren. Daarin kwam naar boven dat er een groot verschil bestaat tussen de commerciële producten voor het publiceren van Linked Data en de toepassingen ontwikkeld door het onderzoeksveld. Een belangrijk resultaat is ook dat performantie van elk van die systemen voor datapublicatie erg afhankelijk is, van zowel de context als de hardware-opstelling. Met de context kan bedoeld worden: de grootte van de dataset, het aantal gelijktijdige dataconsumenten, de effecten van server-cache, en de invloed van het type query. Bij hardware-opstelling

denken we aan de hoeveelheid werkgeheugen van de rekennodes, de mate waarin die optimaal geconfigureerd zijn, alsook de horizontale schaalbaarheid. Ten slotte gebruiken we ook de data van Ontoforce, om na te gaan in hoeverre deze systemen aan hun data- en querybehoeften tegemoetkomen. Daar blijkt dat er nog veel ruimte voor verbetering is.

Ten slotte lijsten we een aantal pistes op, waar verder onderzoek naar gedaan kan worden. Zo bestaan er alternatieve benaderingen om het BLSSpeller-algoritme te parallelliseren, die op het eerste zicht veelbelovend zijn. De inzichten uit het onderzoek naar datapublicatie kunnen aangewend worden om de BLSSpeller data effectief te gaan publiceren als een motiefdatabase en om die te integreren met andere databronnen. Dit zal evenwel gepaard gaan met extra onderzoek naar publicatiemethodes, die zich specifiek op dit type van sequentiedata focussen. De gepubliceerde data kan ook gebruikt worden als input voor nieuwe datamining-algoritmes, bijvoorbeeld om structurele motieven te zoeken, alsook om genmodules te detecteren.

Een algemene conclusie van dit werk is dat het van groot belang is om al bij de start van een onderzoeksproject na te gaan hoe de resultaten gepubliceerd zullen worden. Dit kan immers een grote impact hebben op de manier waarop het onderzoeksveld die resultaten nog verder kan gebruiken.

Summary

The current state of the Life Sciences domain shows a lot of similarities with the state of the domain of physics in the seventeenth century: the successive breakthroughs are proceeding at a dazzling speed. This might even be further sped up by the breakthroughs in Big Data technologies.

The Big Data hype has caused a revival of the research in data management systems. The current focus of this research is in systems that scale horizontally. This corresponds to distributed systems which each manage a fraction of the data instead of a single machine.

Mining for patterns in these Big Data systems, requires the design of novel parallel algorithms, that can exploit the properties of these systems. All this introduces a feedback loop: better tools for data management and more successful algorithmic approaches, will encourage the generation of even more Life Sciences data. Also the rapid evolution of genome sequencing technologies does not hinder this trend: the price for sequencing an entire genome has already dropped to less than $1000.

The Big Data hype was not created from thin air. Successful Internet companies, such as Google and Amazon, have been pioneers of Big Data. Many companies therefore try to copy their Big Data approach. Also in the research field this has not gone unnoticed. It is equally important to invest in (more) data collection as in designing sophisticated prediction algorithms. Big Data algorithms are often conceptually easier than their single-node counterparts. Having them scale is, however, a big challenge in itself. Ideally, we would like to obtain horizontally scalable algorithms, which can process twice the amount of data, with twice the amount of resources, in approximately the same time window.

In part I of this dissertation we explore two types of parallel technologies: MPI (Message Passing Interface) and MapReduce. MPI is a protocol that enables a cluster

of compute nodes to send messages to each other. In the different communication types we mainly distinguish between point-wise communications, where one node A sends a message to node B, and collective communications which require a group effort of the entire cluster. MapReduce is a much simpler paradigm, that initiated the Big Data hype, as Google demonstrated its central role in their data processing architecture. MapReduce is less flexible than MPI, but it takes away many of the responsibilities of the end-user: it takes care of load-balancing, data partitioning and monitoring the health of the different worker-nodes. In this dissertation the loss of flexibility, as compared to MPI, was more than compensated by the other advantages.

In part II of this thesis, two parallel algorithms are discussed, which were developed during this PhD. Both algorithms have the goal of detecting patterns in DNA. The first algorithm, i-ADHoRe, compares the genomes of different organisms, but only focuses on the gene sequences. The algorithm tries to detect collinear regions. These correspond to gene sequences, which show high similarity in between two chromosomes. These regions offer additional insights into the evolution of genomes through time and offer insights into the composition of the genome of the common ancestor of two organisms. i-ADHoRe is especially strong in the detection of whole-genome-duplications. It has been shown that these duplications have played a major role in the survival of species during mass extinction events. The i-ADHoRe algorithm uses MPI to parallelize the workload and can process some of the most challenging datasets in less than an hour. The algorithms also performs better than its competition in terms of sensitivity, by relying on a specially designed multiple sequence aligner. The resulting alignments are used as profiles which are used to scan the gene sequences for weaker patterns, that would otherwise go unnoticed. The method was also biologically validated and it manages to reproduce certain well-established biological results.

The second algorithm operates on the DNA base pair level. BLSSpeller tries to discover short DNA sequences, which we call regulatory sequences. These typically reside in the promoter sequence of a gene. This segment of non-coding DNA precedes the coding gene sequence, but can easily consist of thousands of base pairs. The regulatory elements affect the rate of gene transcription, the rate at which a gene's DNA is converted into RNA. These elements affect gene expression by recruiting transcription factors, which can help with or can obstruct the binding of RNA polymerase. The BLSSpeller algorithm is an exact and exhaustive method

which lists every possible DNA-pattern. Different from its predecessors is the focus on phylogenetically related sequences, such as the Monocot gene families used in this work. The exact algorithm spells all possible DNA patterns, but makes use of a clever branch-and-bound strategy, which drastically limits the pattern search space. Another distinguishing feature of BLSSpeller is, that it evaluates patterns in a genome-wide fashion, by analyzing in how many different gene families a pattern occurs. In terms of parallelization both MPI and MapReduce were used, but only the latter could offer the scalability, to enable the exploration of a sufficiently large pattern space.

In part III of this dissertation we explain how the patterns, discovered in part II, can be published in a cost-effective and scalable way. This can be accomplished using Semantic Web technologies. These technologies enable one to publish data in a decentralized way, while also allowing easy data integration and consumption. These properties are essential in making Life Sciences data reusable. This is a consequence of the interdisciplinary nature of the Life Sciences domain.

In the SEQUEL project, a collaboration with Ontoforce, the current state-of-the art in Semantic Web technologies is assessed. This is done using both artificial and real-world datasets, as used in the back-end of Ontoforce's product. An important result in this benchmark project is that the performance of these systems very much depends on the context and system setup. Context can be: the size of the data, the number of simultaneous data consumers, the effect of server cache, and the type of queries. In terms of setup we distinguish between different amounts of RAM, the effect of proper system configuration and the ability to scale horizontally. Finally, we also studied if these systems could meet the requirements, imposed by Ontoforce's exploratory user interface DISQOVER. Especially for this case there seems to be plenty of room for improvement.

In a final chapter we list a number of new avenues of research which might be worth exploring. There are different parallelization schemes possible for the BLSSpeller algorithm, which at first sight seem promising. The lessons learnt from data publishing can be used to effectively publish the motif database and to integrate it with other (linked) data sources. This will however require additional research, to verify what are the most suitable methods for publishing sequence data. Finally, the published data can also serve as the input for new pattern mining algorithms.

The motif database could be used to discover structured motifs or to detect gene modules.

An important conclusion, which is clearly demonstrated in this work, is the importance of thinking about a data publication strategy, already at the initiation of a research project. This publication will have a large impact on how the research field will re-use the results and should therefore not be treated as an afterthought.

Chapter 1

Introduction

... to boldly go where no one has gone before!

—Capt. Jean-Luc Picard.

In his novel, 'The Island of the Day Before', Umberto Eco takes the reader to the 17th century, to witness the birth of science and to meet some of its fathers, such as sir Isaac Newton. The New York Times review of this book (by Robert Kelly[1]) reads: "*It explores one of the most exciting periods in intellectual history: the mid-17th century, when alchemy and chemistry are still entwined, when Descartes is still slowly articulating his world view, when Galileo for all his prudence can let himself see the moons of Jupiter, precipitating changes not just in our view of the solar system but in the way we reason from appearances.*"

The same atmosphere can be felt amongst researchers in the Life Sciences domain, where successive breakthroughs are proceeding at a dazzling speed. A deep dive in the field of genetics and computational biology is therefore very well described by Captain Picard's quote.

1.1 Motivation

The goal of this dissertation is to design computational approaches which deepen our understanding of the processes governing genome evolution and (transcriptional) gene regulation. These approaches typically use sequence data, corresponding to DNA, RNA or proteins.

[1] https://archive.nytimes.com/www.nytimes.com/books/98/12/06/specials/eco-island.html

This omic data often qualifies as *Big Data*, in this work however the data is not large in volume. *Big Data technologies* are thus used as a means to *distribute* the workload and speed up the processing of the data mining algorithms. Big Data technologies are the subject of part I of this dissertation.

This dissertation will focus on the DNA sequence. The main reason for this is that computational biology is a *data-driven* science, or put even more extreme a 'dataset-driven science': new datasets typically trigger a lot of new research, which is also the case in this work.

The algorithms proposed in part II of this work are *pattern mining* algorithms. The patterns we try to uncover are related to how genomes evolve and to the DNA sequences which affect gene expression, i.e. the rate at which a gene is transcribed.

Research in the *Life Sciences* domain is inherently multi-disciplinary. Therefore, it is important that the different sub-domains can communicate their gained knowledge in an efficient manner. Part III of this dissertation focuses entirely on the *publication of Life Sciences data* and provides insights in the current state-of-the-art of Semantic Web technologies.

An exponential growth of omic data Also in the field of data publication Big Data technologies play a crucial role. GenBank[2], a database with genetic sequences, has been growing exponentially since it's inauguration in 1982. The number of DNA bases in GenBank has doubled approximately every 18 months.[3] As of February 2018, it contains approximately 254 billion bases from 207 million sequences.

This exponential growth can be largely explained by the recent advances in genetic sequencing, called *next-generation sequencing*. One of driving forces behind these breakthroughs was the Human Genome Project[4]. The goal of this project was to unravel the entire Human genome sequence, which is made up of approximately 3 billion base-pairs.

The project kickoff was in 1990 and it took approximately 10 years to come up with a first draft of the human genome. The estimated price would be somewhere between $500 million and $1 billion.

[2] https://www.ncbi.nlm.nih.gov/genbank/
[3] https://www.ncbi.nlm.nih.gov/genbank/statistics/
[4] https://en.wikipedia.org/wiki/Human_Genome_Project

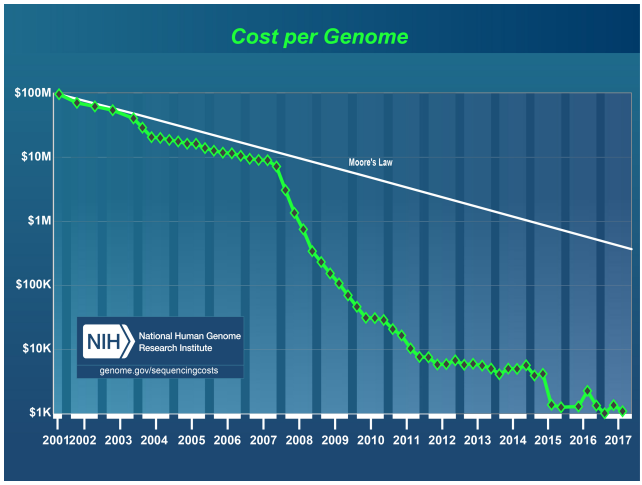


Figure 1.1: The cost for sequencing a genome drops quicker than Moore's Law. The chart starts in the period near the end of the Human Genome project. Since then the price per genome has fallen from $100 million to less than $1,000 in 2017.

As can be seen in Figure 1.1 (from genome.gov[5]), this price has dropped spectacularly, mainly due to emergence of new sequencing technologies, at a rate exceeding Moore's Law. The current price of approximately $ 1000 opens the door for affordable personalized medicine.

Just like the successes of Big Data motivated companies to store and analyze more data, thus creating a feedback loop, the amount of sequence data will skyrocket in the coming years. An example of a company already accumulating sequence data is 23andme[6] which offers DNA analysis services based on a sample of your saliva.

1.2 Dissertation Content

Structure of this book

This book corresponds with two different lines of research which correspond to parts II and III.

In part I we give a broad introduction to Big Data technologies. In Chapter 2 we will introduce two technologies for parallel data analysis: MPI and MapReduce. In

[5] https://www.genome.gov/27565109/the-cost-of-sequencing-a-human-genome/
[6] https://www.23andme.com/

Chapter 3 we introduce the ideas behind the Semantic Web, which consists of a set of technologies for (distributed) data publication.

In part II two bioinformatics problems are tackled with the goal of mining for sequential patterns. In Chapter 4 we provide more insights into the nature of the sequential data by providing basic insights in the biology surrounding DNA and gene sequences. In Chapter 5 we try to detect conserved gene sequences, while in Chapter 6 we dive deeper to the DNA base level where we try to discover conserved sites in promoter sequences.

The common vision in part II is that computational approaches relying on *comparative genomics* have a higher sensitivity to biological patterns. Working with more species at once comes with a price: there is a necessity to move to scalable algorithms that can exploit the combined power of multiple compute nodes in a data center or Big Data platform.

In part III we investigate the ability of Semantic Web technologies to publish, integrate and query Life Sciences data, as is produced by the algorithms in part II. In Chapter 7 we perform benchmarks to evaluate the current ability of these technologies to fulfill the needs of the Life Sciences domain. In chapter 9 we provide final recommendations for the publication of the data in part I.

Main contributors The main supervisors of part II are Prof. Jan Fostier and Prof. Klaas Vandepoele. The main input from pre-doctoral researchers came from Dr. Sebastian Proost in Chapter 5 and Dr. Jan Van De Velde in Chapter 6.

The main supervisors of part II are Prof. Erik Mannens and Prof. Ruben Verborgh. The main input from pre-doctoral researchers came from Dr. Laurens De Vocht. The results in part III are the result of the SEQUEL project[7] in collaboration with Ontoforce, where much input was provided by Dr. Kenny Knecht, Dr. Filip Pattyn and Hans Constandt.

The resulting publications can be found in Appendix A.

[7] The research activities were funded by VLAIO (the Agency for Innovation and Entrepreneurship in Flanders) in an R&D project with Ontoforce, Ghent University and imec.

Research Hypotheses & Research Questions

The topic of this dissertation can be best understood by having a look at the initial research questions and hypotheses:

Hypothesis 1: *A sustained focus on algorithmic optimization and a proper choice of parallelization scheme, is a valid approach for solving big problems (in life sciences) which were previously tackled by relying on heuristics, sampling and approximation methods.*

Hypothesis 2: *Results generated by Big Data algorithms only become valuable when proper attention is given to the way this data will be consumed by the end-user. This requires paying attention to how data will be queried, visualized and integrated.*

RQ1 *Is an approach based on collinear clustering and profile-based detection of homology more sensitive than other approaches and can this be supported by biological evidence?*

RQ2 *Can an exhaustive algorithm be a feasible approach to detect exact patterns in DNA sequences? In other words, what is the size of the search space an exhaustive pattern mining algorithm can manage in a distributed setting?*

RQ3 *What enhancements can lead to a more reproducible and reliable method for benchmarking RDF data management systems?*

RQ4 *What is currently the most cost-effective approach for producing Linked Data on the web and what are the different trade-offs associated with this choice?*

The questions and hypotheses are covered in more detail in the following chapters:

- Chapter 5 deals with **RQ1**.
- Chapter 6 covers **RQ2** and relies heavily on **Hypothesis 1**, since the software was specifically optimized for the problem at hand. A drawback here is that the BLSSpeller code is tailor-made and can therefore only be applied to comparative sequence mining problems.
- Chapter 7 provides answers to **RQ3** and **RQ4**.

- **Hypothesis 2** plays a central role in explaining the shortcomings in the (lack of) data publication strategy in part II and was taken into account when releasing the results of the benchmarking study in Chapter 7.

Part I

Big Data Technologies

Chapter 2

Big Data Processing Frameworks

ARD'RIAN: Good. Then you won't forget me.
DATA: I am incapable of forgetting. I will remember every detail [...] with perfect clarity.

—From 'The Ensigns of Command', Stardate 43133.3

Although the term 'Big Data' was yet to be introduced when the famous fictional character Lieutenant Commander Data[1] appeared in the Star Trek Series (TNG: 1984-1994), Commander Data clearly qualifies as a piece Big Data technology, with a storage capacity of 100 petabytes and a processing capacity of 0.06 PFLOPS[2].

The script writers of the series may however not have been familiar with Moore's law [13], since today's supercomputers already exceed this capacity by far. The number one supercomputer has its record performance at 93.015 PFLOPS and Big Data companies such as Facebook already store data in the same order of magnitude: Facebook stores approximately 357 petabytes of photos as of 2017. However, to make the comparison fair, we must stress that we are comparing a data center's abilities to that of a single 100 kilogram android.

What is Big Data? The Oxford English Dictionary gives the following definition: "Data of very *large* size, typically to the extent that its *manipulation* and *management* present significant logistical *challenges*."

[1] Source: `http://memory-alpha.wikia.com/wiki/Data`
[2] PFLOPS: 10^{15} floating point operations per second.

Focusing on size alone is however an oversimplification. The challenges related to Big Data are often described using a number of Vs: *Volume*, *Variety* (data formats), *Velocity* (streaming data) and *Veracity* (data quality).

If Big Data is so challenging then why bother? For the answer to this question two papers are often cited. The first one is coauthored by Google's Research Director Peter Norvig [12], titled: "The Unreasonable Effectiveness of Data". In the abstract the authors suggest that for many problems it might not be possible to capture the complexity using simple formulas such as Newton's Second Law. Instead, given a lot of data, new algorithms can be used to build high-quality models. Dr. Jim Gray (Microsoft) went as far as calling it a scientific paradigm shift[7]: moving from a hypothesis-first to a *data-first* approach.

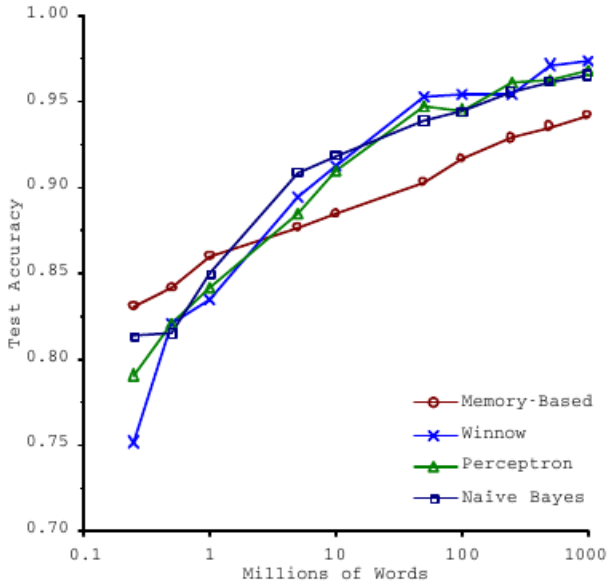


Figure 2.1: Learning rate for different algorithms on a task of confusion set disambiguation From Banko and Brill [1]

The second influential paper is that of Banko and Brill [1]. The authors ran an experiment where they compared different algorithms for *confusion set disambiguation*: the algorithms were tested for their ability to guess the missing word in a text, given a set of commonly confused words to choose from.

A famous figure from this paper is Figure 2.1 where they compare the learning curves of different algorithms, i.e. their accuracy as a function of the size of the training corpus. The figure shows that although the algorithms under test are very different, their performance is very similar for large corpora.

Additional observations:

1. Algorithms which are top-ranked for small corpora (left of the figure) might be worse when the amount of training data increases.

2. Increasing the amount of training data leads to a monotonous increase in accuracy.

All of the above motivates the adoption of a new type of technologies which can fully exploit Big Data. These *Big Data technologies* are the subject of this part of the dissertation.

How do we interact with (Big) Data? The Crisp-DM methodology [15] is the most widely accepted method for describing the data science process. However, in the context of Big Data, a more complete picture can be seen in Figure 2.2, which shows additional components corresponding to interactive visualization and knowledge management. To perform these interactions typically Big Data technologies are required.

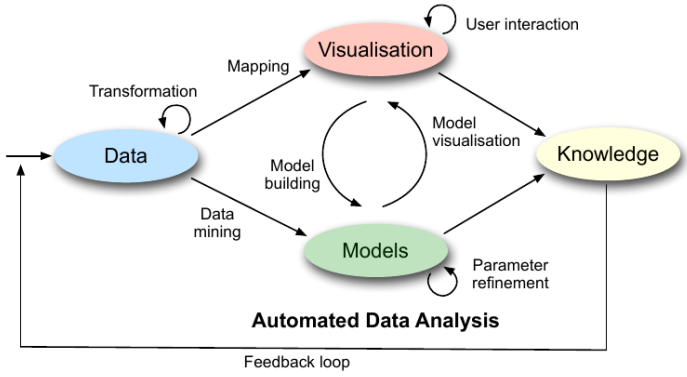


Figure 2.2: Different types of interactions in the (big) data analysis process. Figure from Ellis[4].

The first problem technology has to solve is how to *store* and how to *process* the data. These technologies play a role in the arrows for data transformation, mapping, and mining.

Data can be stored in raw files or in database management systems, which impose a certain *schema* on the data. *Distributed file systems*, such as the Hadoop Distributed File System (HDFS), are a common approach for storing raw data in both a scalable and a fault-tolerant way. In order to allow more complex interactions with data, for example via querying, data can also be stored in a range of different *(No)SQL storage solutions*, often with their own query language. In chapter 7 these systems will be discussed in the context of semantic databases.

R and Python are two typical scripting languages with a variety of packages for transforming data and training models. Although single-node boundaries can be pushed, by supporting out-of-core learning (see for example scikit-learn strategies for bigger data[3]), there comes a point when data has to be stored in a distributed system and processed in parallel. *Big Data processing frameworks* such as Hadoop and Spark both have libraries for distributed data transformations and scalable machine learning. Hadoop and the MapReduce framework will be introduced in section 2.2.

Prior to the rise of Big Data platforms, different paradigms for *parallel programming* were already in use.

- Shared Memory Architecture: Parallelization is obtained via *Multithreading*, where a number of processes share a memory pool.
- Distributed Memory Architecture (Figure 2.3): Each process has its own memory and processes exchange information via *Message Passing* using a fast interconnection network (Infiniband).

Complex (high dimensional) datasets are often difficult to *explore* and *visualize*. This can be addressed by trying to map the data onto a lower dimensional space using linear *dimensionality reduction* techniques such as PCA or nonlinear techniques generally called *manifold learning*. Another approach is to use tools that bring the *human-in-the-loop* via user interaction. Examples of such an approach are tools for interactive data visualizations. A user then typically gets a number of handles to zoom, filter, and query to get different views on the data.

[3] `http://scikit-learn.org/stable/modules/scaling_strategies.html`

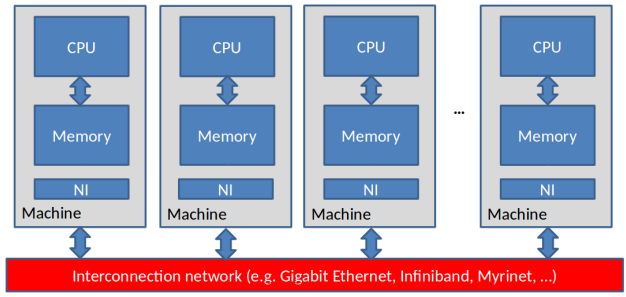


Figure 2.3: Distributed Memory Architecture (Figure from [2])

The results of the False Discovery Rate analysis of the BLSSpeller algorithm in chapter 6 depend on 6 parameters and can be explored in an interactive web page. In chapter 7 we designed a feature matrix and a scoring table to allow the end-user to rank different knowledge management systems according to a custom set of selected and weighted criteria.

Chapter 3 on Semantics will provide an answer on the technologies needed on the right-hand side of Figure 2.2. There we will delve deeper into handling the resulting *Knowledge* of the data science process. Essentially there is to enable knowledge integration and data re-use.

Big Data Technologies in this work

The initial efforts towards parallelization in this work focused on using *MPI* (Message Passing Interface) with heavily optimized source code in C++. The first version of the BLSSpeller algorithm turned out to have scalability issues. Analysis showed that this could be attributed to: (i) a lack of fault tolerance, (ii) no dynamic scheduling, (iii) a memory-only approach. To address these issues, the algorithm was rewritten from scratch (to Java) and the *Hadoop* (v1) *MapReduce* framework was used for parallelization. This framework automatically provides fault-tolerance and dynamic scheduling. Also, it relies on secondary storage, thus circumventing the limitations imposed by memory.

For post-processing we made use of *Apache Pig*[4]. This is a tool which generates MapReduce code, while using a much less verbose relational scripting language, called Pig Latin.

[4] `https://pig.apache.org/`

In chapter 7 many *NoSQL technologies* will be discussed, including an approach which relies on the MapReduce successor *Apache Spark*[5].

In this work *D3.js*[6] is used for interactive in-browser exploration of the results of the BLSSpeller algorithm in chapter 6.

Big Data Infrastructure in this work

The initial experiments in this doctoral thesis, in chapter 5, were run on the *Tier-2* Stevin supercomputing infrastructure of Ghent University. This Tier-2 system, named 'gengar' consisted of 192 nodes, each with 16 GB RAM and 8 cores per node (Intel Xeon Harpertown), all connected via an Infiniband network (DDR). Gengar has been decommissioned as of 2014.

The results of the Hadoop simulations in chapter 6 were obtained using the cloud infrastructure of *Amazon Web Services*. More specifically using the Amazon *S3* storage service in combination with the *EC2* service (elastic compute cloud) as the hardware provider and the *EMR* service (Elastic MapReduce) to run Hadoop. The clusters consisted of 20 nodes of the type `m1.xlarge`, with 4 vCPUs per node, 15 GB RAM, 4 x 420 GB disk drives (I/O parallelization) and a high network performance.

Later Hadoop simulations, run during the revision process of the BLSSpeller paper, were run on the *Tier 1* Stevin supercomputer. This is a cluster which consists of 528 nodes, each with 64GB RAM and 2×8 cores per node (Intel Xeon E5-2670 - 2.6 GHz). This sums up to a total compute power of 152.8 TFLOPS/s and a total 33 TB of RAM. Nodes are connected via an Infiniband network (FDR Infiniband Mellanox interconnect).

An important distinction between the two infrastructures is the very *different approach to secondary storage*. For the simulations on AWS every node in the Hadoop Cluster has only access to its own *local* hard disk drive(s). On the Stevin supercomputer all nodes have access to a *shared* set of RAID disks with a Storage Area Network (SAN). This has an important impact on the inter-node communication (between the map and the reduce phase). In the regular setup where *HDFS* is used, the communication phase consists of the data being written to disk and sorted within every node. This is followed by a shuffle phase where all data is sent over the network to the correct reducer node for the final computations. On the supercomputer the

[5] `https://spark.apache.org/`
[6] `https://d3js.org/`

shared storage can be exploited by using a different distributed file system: *Lustre*[7]. In most cases this can lead to performance gains of up to 30% on typical Hadoop Benchmarks (Terasort Benchmark). A more in depth review of this system and its performance can be found in the Lustre wiki[8].

The benchmarking efforts of RDF databases in chapter 7 were performed on different types of AWS EC2 servers and the imec iLab.t infrastructure (Virtual Wall[9]).

2.1 Message Passing Interface

The Message Passing Interface (MPI) is a protocol for parallel computing. MPI is the de facto standard for communication in High Performance Computing (HPC). Different implementations exist, the HPC infrastructure of Ghent University for example supports Intel MPI[10].

MPI is more general than Big Data technology The goals of parallel computing are broader than that of Big Data technologies. The latter have as their primary goal to process big datasets in a data-parallel fashion, i.e. every process executes the same task on a different subset of the data. According to Flynn's taxonomy [5] this type of parallelism is called 'Single Instruction Multiple Data' (SIMD). Although MPI supports this type of parallelism, it is more general in that not all nodes have to perform the same set of instructions at the same time. This is called 'Multiple Instruction Multiple Data' (MIMD). One drawback of MPI is that there are currently only bindings for C, C++, and Fortran. Furthermore, the flexibility of the MPI model also gives a lot of responsibility to the end-user. We will revisit this argument in the section 2.2.

MPI is essentially a collection of routines for inter-node communication, the most important being point-to-point communications between 2 processes and collective communications between 2 or more processes. Other functionality includes defining cluster topologies, synchronization, and parallel I/O.

Running an MPI program means running the exact same code on P independent processes. This execution can be customized per processes due to the `MPI_Comm_Rank`

[7] `https://en.wikipedia.org/wiki/Lustre_(file_system)`
[8] `http://wiki.lustre.org/index.php/Running_Hadoop_with_Lustre`
[9] `https://www.ugent.be/ea/idlab/en/research/research-infrastructure/virtual-wall.htm`
[10] `https://software.intel.com/en-us/intel-mpi-library`

routine, which assigns a unique (within a communicator) rank r to each process. Every node thus gets an identifier in a range from $r = 0$ to $r = P - 1$.

> **Example:** In a Master-Slave implementation the source code would contain conditionals for testing if the actual process is the master ($r = 0$) or a slave ($r > 0$).

Processes are grouped via *Communicator objects*. `MPI_COMM_WORLD` is the general group for all processes in a simulation. These communicators determine between which processes messages need to be shared, during collective communications. The communicators also take into account the network topology to optimize communications.

Let's have a closer look at the different communication routines:

Point-to-Point Communications In Point-to-Point communications one process ($r = a$) sends a message to a second process ($r = b$). The code of process 'a' contains an `MPI_Send` routine which defines the message and the rank of the destination process 'b'. Process 'b' then has an `MPI_Recv` with the same metadata about the message and a source rank a from which the message is to be received. Note that this communication is *blocking* (handshake protocol), i.e. if 'a' does not send the message, 'b' will never proceed.

The blocking communication can lead to *deadlocks*, where both processes are waiting for input from each other. Furthermore blocking communication can lead to idle time.

The first issue can be circumvented by the `MPI_Sendrecv`, which allows the *exchanging* of messages and is thus less sensitive to deadlocks.

The second issue can be tackled by *asynchronous* (non-blocking) communication routines. In the latter, the process can perform local calculations, while sending and receiving messages. This is typically implemented by periodically polling a *status* object, which is assigned to a communication routine.

Collective Communications Most relevant to this dissertation, are the routines for collective communications. In this type of routines all processes in the communicator are involved. We distinguish between the following types:

- One-to-All: (i) `MPI_Broadcast` sends a message from a single root process to all the other processes in the communicator; (ii) `MPI_Scatter` takes an array of P

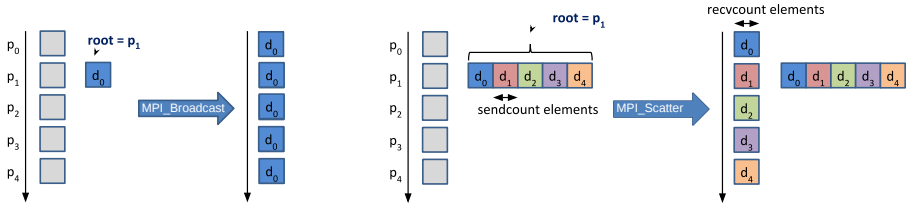


Figure 2.4: One-to-All Operators: Left: `MPI_Broadcast` replicates a piece of data (d_0) to the other nodes in the communicator. Right: In `MPI_Scatter` an array is broadcasted such that every node receives a different element d_i. Figure adapted from [2].

messages and sends each element with index i to a process with $r = i$. This is visually depicted in Figure 2.4.

- All-to-One: (i) `MPI_Gather` is the inverse operator for `MPI_Scatter`; (ii) `MPI_Reduce` is a collective computation on the data in the different processes. The computation has to be an associative operator, such as `MPI_SUM` or `MPI_MAX`, since the order of operations is determined by the communicator. This is visually depicted in Figure 2.5.
- All-to-All: Just as with the one-to-all operators, a distinction can be made between operators for which all receivers get the same data or for which they get a unique slice. `MPI_AllGather` and `MPI_AllReduce` are a combination of the All-to-One operators followed by a broadcast operation. In `MPI_Alltoall` and `MPI_Reduce_scatter` the same happens but the broadcast operator is replaced by a scatter operator. This is visually depicted in Figure 2.6.
- Synchronization: All processes wait until the last process has called `MPI_Barrier`.

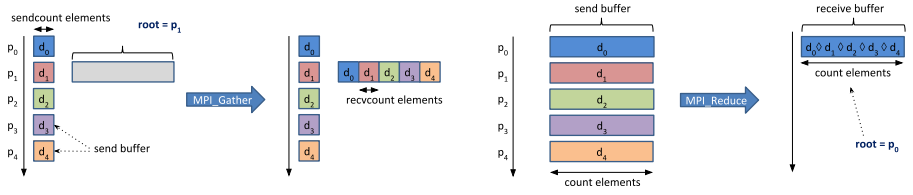


Figure 2.5: All-to-One Operators: Left: `MPI_Gather` sends all elements in the different processes to a root process. Right: `MPI_Reduce` Also gather all data in one root process while performing an associative operation $\diamond$. Figure adapted from [2].

In this PhD we used the vectorized versions of the routines described earlier: `MPI_AllGatherv` and `MPI_AlltoAllv`. The rationale behind the communications remains

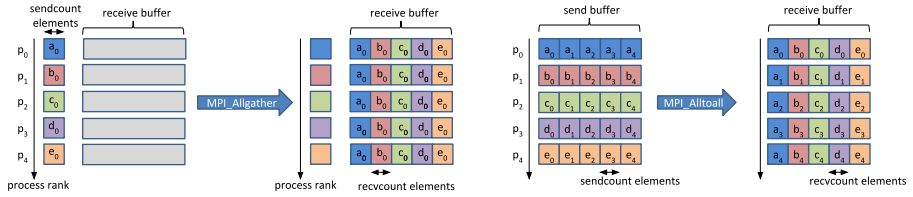


Figure 2.6: All-to-All Operators: Left: `MPI_AllGather` is equivalent to a gather operation followed by a broadcast operation. Right: `MPI_Alltoall` resembles a matrix transposition and corresponds to a gather operation followed to a scatter operation. Figure adapted from [2].

the same, but the number of elements sent to each process can vary. This is taken care of by specifying the offsets in the send buffer. Other operators are `MPI_Broadcast` and `MPI_Comm_rank`. The later is used for identifying the different process ranks. Finally `MPI_Pack` was used to store an array of messages into a memory efficient representation before sending it to another process.

Performance optimizations of collective operators An important distinguishing feature of MPI, as opposed to Big Data processing frameworks, is the way in which these collective operations are optimized.

To elaborate on this topic we introduce some notation. The time to send n bytes in a point-to-point communication is given by:

$$T_{comm} = \alpha + \beta \cdot n$$

with α a fixed latency, and the bandwidth $B = \frac{1}{\beta}$ (bytes/sec). P is used to denote the number of processes in a compute cluster.

A naive implementation of the `MPI_Broadcast` routine is expected to take $T_{comm} \cdot P$ time. The binary tree implementation, depicted in Figure 2.7, however can speed this up to $T_{comm} \cdot \log_2 P$ time.

For `MPI_Scatter` and `MPI_Gather` the data for each receiver or the data sent respectively, is unique so the fraction $\beta \cdot n \cdot P$ cannot be reduced, the latency can however be reduced from $\alpha \cdot P$ in a naive implementation to $\alpha \cdot \log_2 P$ as can be seen in Figure 2.8. The same optimization can be achieved for the `MPI_reduce` operation.

The `MPI_AllGather` is equivalent to P calls to the `MPI_Gather` routine, but the fact that all processes end up with the same data again allows for a clever optimization. Using the *butterfly algorithm*, depicted in Figure 2.9, again a latency of $\alpha \cdot \log_2 P$ can

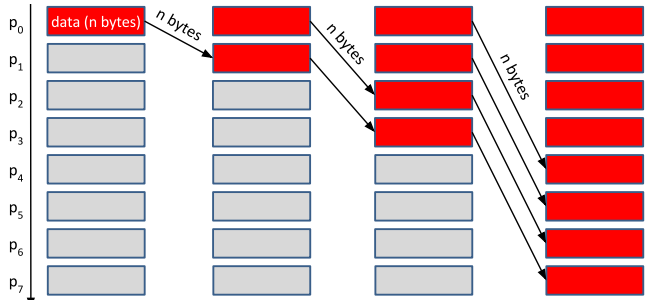


Figure 2.7: Binary tree broadcast algorithm only takes $T_{comm} \cdot \log_2 P$ time.

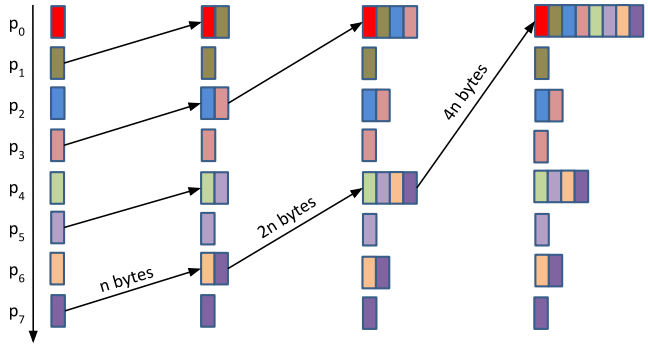


Figure 2.8: `MPI_Gather` binary algorithm: The number of communication rounds can be reduced from P to $\log_2 P$. Note that the algorithm does not reduce the amount of data sent to the root process, since the individual elements are all unique.

be obtained. This is more efficient than the combination of the individual gather operator followed by a broadcast.

The `MPI_Alltoall` operator cannot take advantage of this binary tree approach and requires $P \cdot T_{comm}$ to send the data.

The main source for this section is the course on Parallel and Distributed Systems of Ghent University [2].

2.2 Hadoop MapReduce

Inverted index of the Web An *Inverted Index* is a data structure used in full-text search engines. It is used to quickly find documents which contain a certain search term. Figure 2.10 shows an inverted index[11] in a toy example with 4 documents

[11] Figure from: https://www.slideshare.net/erikhatcher/introduction-to-solr-9213241

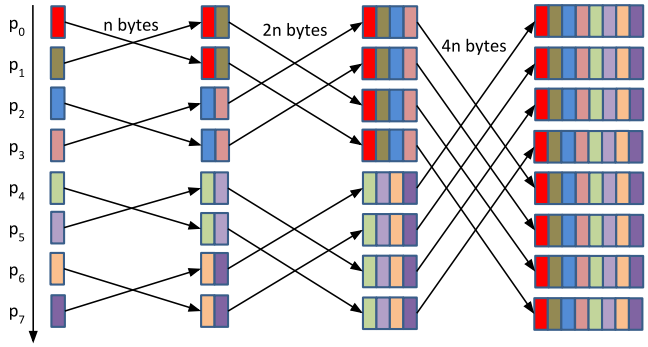


Figure 2.9: Butterfly algorithm for `MPI_AllGather` again reduces the number of communication rounds to $\log_2 P$

each containing a single line of text. The inverted index is a sorted map (on disk), with the terms as the keys and *posting lists* as the values. In the example the posting lists only contain the document identifier but in real-world search systems, of which Google Search is the most commonly used, these posting will also contain positional information, for example whether a term occurs in the title of a document.

Building an inverted index for the entire world-wide web (WWW), is a challenging task. It is a true Big Data problem, as the size of the inverted index is of the same order as the entire WWW. Google published two papers, which revealed the principles behind the distributed data systems they designed to build and maintain their inverted index:

- In 2003 they described the *Google File System* [6], a distributed storage layer with fault-tolerance and scalability built in by design.
- In 2004 they described *MapReduce* [3], which is a framework for performing data-parallel computations in a scalable way, with the main feature that data locality is exploited: "*Run the computations where the data is*".

The Apache Nutch project, which tried to build an open-source infrastructure to index the WWW, picked up both papers and created an open source implementation, which they coined *Apache Hadoop*. In 2008 Hadoop started gaining traction, they broke the record on the Terasort benchmark. Hadoop managed to sort 1 TB of text in 3.5 minutes [11] using a cluster of 910 nodes, each with 8GB of RAM and 4 dual cores. This record is important as it tests only the framework itself. This is because

ID	Text
1	Baseball is played during summer months.
2	Summer is the time for picnics here.
3	Months later we found out why.
4	Why is summer so hot here

↑ Sample document data

Dictionary and posting lists →

Term	Freq	Document ids
baseball	1	[1]
during	1	[1]
found	1	[3]
here	2	[2], [4]
hot	1	[4]
is	3	[1], [2], [4]
months	2	[1], [3]
summer	3	[1], [2], [4]
the	1	[2]
why	2	[3], [4]

Figure 2.10: Left: 4 sample documents with some text. Right: The inverted index, which stores for each term a linked list of document identifiers (= posting list) as well as the number of documents in which the term occurs.

MapReduce, without any customized code, is in fact a framework for sorting key-value pairs, therefore the Terasort Benchmark measures the ability of the framework at performing a distributed *Merge Sort*.

What is Hadoop? A single-sentence summary of Hadoop could read: *Hadoop is a scalable fault-tolerant Big Data operating system.* One of the main ideas behind Hadoop is the observation that for large data clusters, '*Failure is the rule, not the exception*'. As a rule of thumb, one could argue that if a single server fails once every 3 years, then a cluster of 1,000 servers will on average fail on a daily basis [9]. To put this in perspective, the actual amount of servers running at Google was already in 2011 estimated to be roughly 900,000[12]. This corresponds to approximately 1 failure every 100 seconds.

The original implementation of Hadoop has a two-layered architecture. A compute layer (MapReduce) and a storage layer (HDFS), shown in Figure 2.11[13]

- **MapReduce:** The JobTracker splits a job in Map and Reduce tasks which are scheduled on TaskTrackers. The JobTracker communicates with the NameNode to infer the ideal location to execute the (preferably data-local) computation.
- **Hadoop Distributed File System (HDFS):** The NameNode serves as the master and keeps track of where the different file blocks and their replica are

[12] http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers

[13] http://aletheconsulting.com/assets/hadop.png

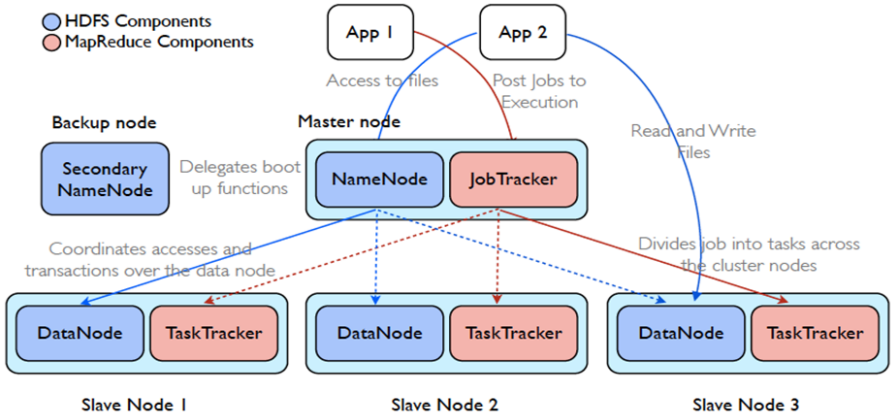


Figure 2.11: Hadoop is a two-layered master-slave architecture. The storage layer, called HDFS, consists of a NameNode (master) and a number of DataNodes (slaves). For the compute layer, called MapReduce, the JobTracker (master) supervises a set of TaskTrackers (slaves).

stored (Figure 2.12 from Holmes [8]). The DataNodes store the actual file blocks and report their health to the NameNode using heartbeats.

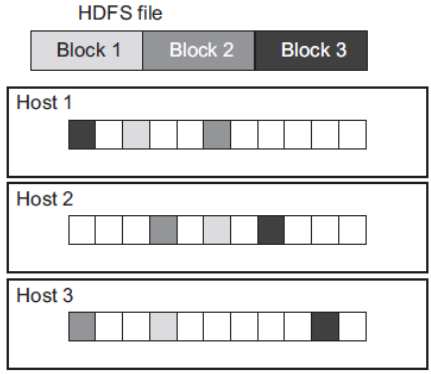


Figure 2.12: HDFS fault-tolerance and scalability is accomplished by partitioning (large) files into file blocks and replicating these to at least 3 DataNodes. The file and block mappings are stored in the NameNode.

The MapReduce framework does not expose any of the challenges related the parallelization to the software developer. Only a Map and a Reduce function have to be implemented. Both of which convert a set of key-value pairs into another set of key-value pairs. This is one of the main differences with MPI, *the MapReduce*

framework takes care of all the tasks below while in an MPI setting these are all the responsibility of the programmer:

- **Partitioning** of the data, such that the code runs 'where the data is'.
- The master-slave setup allows for **dynamic scheduling** as the master is continuously polling for the health and task progress of the worker nodes.
- The standard policy for dealing with **node failures** is that a JobTracker resubmits failed tasks to a different TaskTracker. The following policy is used: If a TaskTracker fails 4 different tasks it is removed from the resource pool, if a task fails 4 times this leads to job failure and finally if a task takes longer than is expected (= straggler) it is executed in parallel on a different machine (speculative execution).

Basic MapReduce example The most basic MapReduce program is the Word Count algorithm, shown in Figure 2.13 (from Berkeley's introductory course on Spark[14]). Note that this is very similar to the task of building an inverted index! (for an in-depth discussion see Lin [10])

MPI Word Count Implementing the same in MPI would correspond to a `MPI_Reduce` for every key followed by a `MPI_Scatter`. However, MPI has a routine to execute this in one step, which allows for optimizations behind the scenes: `MPI_Reduce_scatter`.

Finally, it is important to keep in mind that MPI supports a much wider range of communication routines, instead MapReduce is clearly limited to SIMD. Clearly, not every parallel algorithm can be written to fit the MapReduce paradigm.

In chapter 6 we will see that the BLSSpeller algorithm is in fact rather similar to building an inverted index. In chapter 3 we will see that MapReduce is also capable of resolving SQL and SPARQL queries.

MapReduce data flow To conclude this section we will have a look at the data flow in the MapReduce framework. Contrary to MPI, where we try to keep everything in memory, MapReduce makes extensive use of secondary storage, with its ability to 'spill-to disk'. The MapReduce data flow is shown in Figure 2.14 (from White [14])

[14] https://www.edx.org/course/introduction-apache-spark-uc-berkeleyx-cs105x

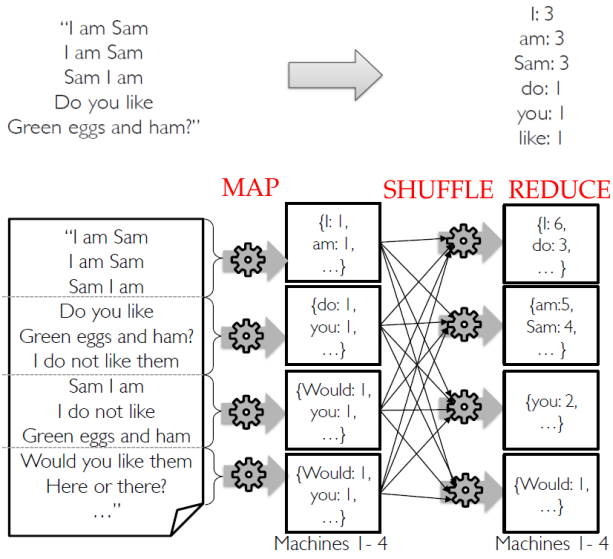


Figure 2.13: The base example: Word Count in with MapReduce.
Top: The goal of the Word Count algorithm is to calculate all individual word frequencies.
Bottom: For large files, which are partitioned in HDFS, Word Count is run in the mappers of the individual nodes. The partial results are key-value pairs (term, partial count). The key-value pairs are shuffled across the network, such that all pairs with the same key end up in the same reducer, where the partial counts are aggregated.

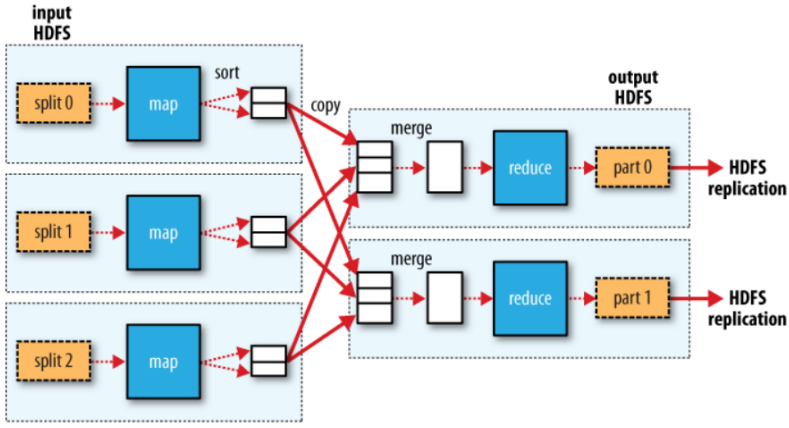


Figure 2.14: MapReduce data flow with multiple reduce tasks. For every unit of work (a split) a Mapper executes map functions which result in a set of new key-value pairs. These key-value pairs are partitioned and sorted in the *sort phase*. In the *copy phase* the files with the same set of keys are sent to the corresponding reducer nodes. There, the individually sorted files are finally *merged*, before the reduce function is called.

The data flow is automatically taken care of by the MapReduce framework. During the *map phase* the mappers emit key-value pairs. Initially, these are stored in an in-memory circular buffer. When this buffer reaches a certain configurable threshold size a background thread will start to spill the buffer to disk. This background thread partitions the data according to the reducer by which certain keys will be processed. Within each partition the key-value pairs are sorted by key. Each spill results in a separate file, but before the actual shuffling begins the individual files are aggregated. Note that both the sorting and the copying over the network can be performed while still in the map phase. Especially in cases where much data needs to be shuffled, as is the case for BLSSpeller, this can lead to a large performance gain.

A reducer gets per key a set of sorted files from the different mappers. Finally, when all map output has been copied to the reducers, the *merge phase* can start. The sorted files are merged (this is in fact a distributed mergesort algorithm). When all data is sorted the *reduce phase* starts with a call to the reduce function for every unique key and all output is written to HDFS.

As a reference for this section we used Tom White's Hadoop Reference [14] and Alex Holmes' Hadoop in practice [8].

References

[1] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. *Proceedings of the 39th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 26–33, 2001.

[2] F. De Turck and J. Fostier. Lecture notes and slides of the course: parallel and distributed software systems. 2017.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Operating System Design and Implementation*, pages 137–150, 2004.

[4] G. Ellis and F. Mansmann. Mastering the information age solving problems with visual analytics. *Eurographics*. Volume 2 of pages 5, 2010.

[5] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system, 2003.

[7] T. Hey, S. Tansley, K. M. Tolle, et al. The fourth paradigm: data-intensive scientific discovery. Volume 1 of Microsoft research Redmond, WA, 2009.

[8] A. Holmes. Hadoop in practice. Manning Publications Co., 2012.

[9] J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of massive datasets. Cambridge university press, 2014.

[10] J. Lin and C. Dyer. Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[11] O. O'Malley. Terabyte sort on apache hadoop. *URL: http://sortbenchmark. org/Yahoo-Hadoop.pdf*, 2008.

[12] F. Pereira, P. Norvig, and A. Halevy. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[13] R. R. Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[14] T. White. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012.

[15] R. Wirth and J. Hipp. Crisp-dm: towards a standard process model for data mining. *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. Citeseer, pages 29–39, 2000.

Chapter 3

Publishing Big Data on the Semantic Web

Colorless green ideas sleep furiously.

—Noam Chomsky

"Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like 'Huardest gefburn'? Kjift - not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look."

Both the quote of Noam Chomsky and the output of the Latex-command 'blindtext' (previous paragraph) are examples of natural language which is syntactically correct but without any meaning. To a machine, all language looks like this, it sees a piece of text from which it can try to learn some features, such as the frequency distribution of the words, a corpus,... but there is something missing: *semantics*. While *syntax* describes the rules by which words can be combined into sentences, semantics describe what they mean[1]. In a nutshell, the *meaning* of a word can only be fully understood by analyzing the *context* in which it occurs and its *relationships* to other words.

[1] `https://dictionary.cambridge.org/dictionary/english/semantics`

Figure 2.2, in the previous chapter, listed the main interactions with Big Data. One thing has not been addressed so far: what to do with the resulting knowledge and how to create a feedback loop? In Figure 3.1 we zoom into this missing piece and address *knowledge publication*.

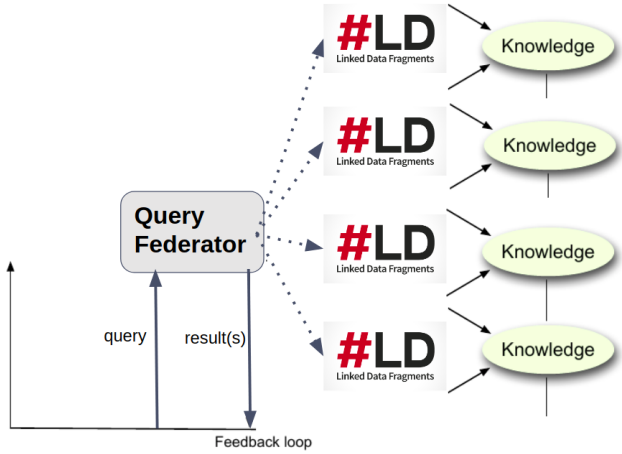


Figure 3.1: Knowledge on the Web can be consumed by making use of a Query Federator which decomposes a general question into subquestions that can be answered by an individual dataset. A published dataset of Linked Data we call a 'Linked Data Fragment'.

This chapter will deal with the challenges related to publishing knowledge. We will specifically focus on publishing knowledge as Linked Data and the associated *Semantic Web technologies* which enable this. In chapter 7 we will explain how this is relevant for the data published in the Life Sciences domain.

In section 3.1 we will introduce the concept of Linked Data and the technology stack enabling the associated 'Semantic Web Vision'. Section 3.2 will describe the Linked Data Life cycle: the iterative procedure required to prepare raw data for publishing. The underlying systems and query language for consuming Linked Data is discussed in section 3.3. Linked Data consumption on the Big Data scale is the subject of section 3.4.

3.1 Linked Data & The Semantic Web: Principles and Technologies

Evolving towards the Web of Data

Since its inception in 1989 the World Wide Web (WWW) has been a tremendous success story with over 1.3 billion websites online right now[2]. The Web has evolved from a read-only system of static pages built on top of HTML, CSS, and javascript, to a more dynamic read-write system coined Web 2.0, where the end-user can add his own content without experiencing technical hurdles. Most notable examples are social media and web blogs.

Semantic Web vision In 2001, in an article in The Scientific American, the founding father of the WWW Tim Berners-Lee, gave his vision for the next stage of the Web [5]. He described the 'Semantic Web', an extension to the current Web with embedded semantics. This Web would enable intelligent software agents to autonomously find and integrate information, while solving complex tasks. The goal of the Semantic Web is therefore to serve as a *global distributed database* of machine-interpretable knowledge. In this context the Web is no longer a 'web of documents', but more generally a 'web of data'.

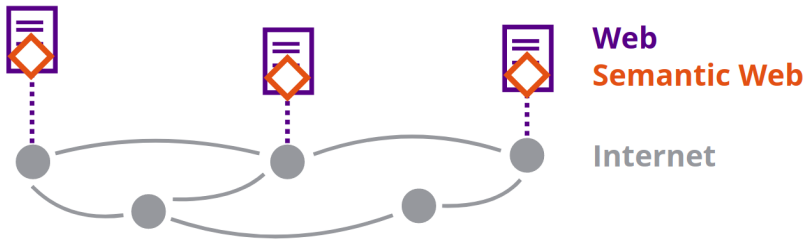


Figure 3.2: The Semantic Web layer is integrated in the current web of documents. One possible approach would be the use of RDFa, which specifies how to embed semantics in HTML tags.

The Semantic Web is not a replacement of the Web 2.0, but an additional layer, as can be seen in Figure 3.2 (taken from the course Web Fundamentals[3]).

[2] February 2018: `http://www.internetlivestats.com/total-number-of-websites/`
[3] `http://rubenverborgh.github.io/WebFundamentals/birds-eye-view/`

Standards To bring the Semantic Web vision into practice, a basic stack of technologies has to be globally agreed upon. This is where the W3C (World Wide Web Consortium), the international organization working to develop web standards, comes into play. W3C came up with a number of Semantic Web standards, most notably RDF, SPARQL, and OWL. We will discuss these in more detail in section 3.1.

Centralized vs Decentralized It might also be interesting to note that the use of semantics on the Web has not gone unnoticed by the current 'forces' governing the Web such as Google. As an example Google is known to use semantics in its Knowledge Graph[4] which is a graph database originally built from Wikipedia, the CIA World Factbook[5], and the Freebase project[6]. In 2016 the Knowledge Graph contained over 70 billion facts and was used to answer 'roughly 1/3 of the Google search queries. An important difference with the Semantic Web vision however, is that the latter stays loyal to the decentralized vision of the WWW, contrary to Google which manages the information in a centralized way only accessible via a (limited) Web API.

Linked (Open) Data

The Semantic Web encompasses all semantically annotated datasets on the Web. Individual datasets we call *Linked* Datasets. The adjective 'Linked' explicitly refers to data not being locked in silos. In his famous Ted Talk in 2009, titled 'The next Web', Tim Berners-Lee describes the current practice of today's major Web 2.0 actors to lock away data, preventing interoperability of datasets. Linked Data, on the other hand, has interoperability built in by design: datasets on the Semantic Web refer to each other, thus effectively connecting them together into a supergraph, thereby reusing and linking resources.

Linked Data principles Four rules lie at the basis of Linked Data and were introduced in the 'Linked Data' note by Tim Berners-Lee[7]:

1. Use URIs as names for things.
2. Use HTTP URIs, so that people can look up those names.

[4] https://en.wikipedia.org/wiki/Knowledge_Graph
[5] https://www.cia.gov/library/publications/the-world-factbook/
[6] https://www.wikidata.org/wiki/Wikidata:WikiProject_Freebase
[7] https://www.w3.org/DesignIssues/LinkedData.html

3. When someone looks up a URI, provide useful information, using the standards.
4. Include links to other URIs, so that they can discover more things.

Using URIs (Uniform Resource Identifiers) to identify resources takes away any ambiguity. Making URIs *dereferenceable*, allows any agent to look up a certain resource, and more importantly find information which is linked to this resource. The latter enables further data exploration. Having the additional information in a standardized format gives software agents the ability to autonomously explore, contrary to the common practice of putting unstructured information in web documents with only human consumption in mind.

Following these principles, solves two of the requirements we put forward in the introduction: the need for semantics and data integration. In the definition of semantics we emphasized that a concept derives its meaning from its 'relationship' with other concepts. For Linked Data this is materialized using typed links. These same links also cross the boundaries of individual datasets, which addresses the integration issue.

Running example As an example we focus on the author of this chapter's quote, Noam Chomsky. On Wikidata the author is identified with the following URI:

`http://www.wikidata.org/entity/Q9049`

If we dereference this URI with our web browser, we get a *representation* of this resource. This page shows us all information which is linked to this URI resource. Some of the links point to other datasets containing possibly complementary information. For example there is a link to the Goodreads author ID:

`https://www.goodreads.com/author/show/2476`

The type or class of the link is also described by a URI:

`http://www.wikidata.org/prop/direct/P2963`

As anyone can create Linked Data, reusing URIs is - although encouraged - not enforced. As an example the DBPedia Linked Dataset also has a resource identifier for Noam Chomsky:

`http://dbpedia.org/resource/Noam_Chomsky`

Open-World assumption The fact that anyone can create information, also has implications on the interpretation of results from information retrieval on the Semantic Web. Inference in regular knowledge systems implicitly assumes a *Closed-world assumption* (CWA), in layman's terms: all information is contained in the knowledge base and a fact can only be true or false. For the Semantic Web the *Open-world assumption* (OWA) holds, which implies that a result also can be unknown. An in-depth coverage of this topic can be found in the book by Reiter [27]. An example to clarify: If we explore the following linked document for Noam Chomsky:

```
http://dbpedia.org/page/Noam_Chomsky
```

We will find all sorts of statements describing this individual, but there is no explicit mention of the fact that he is human. Under the CWA this implies, that Noam Chomsky is not a human. Under the OWA however, the statement is less strong: unknown. This is because we are not guaranteed, to have all the information available, and in fact, if we check the Wikidata document for Noam Chomsky we find (URIs omitted):

```
http://www.wikidata.org/entity/Q9049  instance of  human
```

Linked Open Data The goals of the Semantic Web community very closely resemble that of the Open Data movement. The goal of this movement is to make data freely available, without any legal restrictions. To support the Open Data movement, Tim Berners-Lee suggested a 5-star deployment scheme for Open Data[8], as depicted in Figure 3.3 (taken from 5stardata.info[9]).

The Linking Open Data Project [6] launched in January 2007 with the aim of addressing the Catch-22 of the Semantic Web: Developers have no incentive to build applications on top of Semantic Web technologies, but data publishers don't see the added value of publishing Linked Data as there are no applications to benefit from it. This project kick-started the Linked Open Data (LOD) cloud, a collection of coupled Linked Datasets. Figure 3.4 shows the LOD cloud diagram as of May 2007, with only 12 interlinked datasets. The full LOD cloud in 2007 already consists of over 1 billion statements of which 15 thousand connect one dataset to another. Some of the core

[8] https://www.w3.org/DesignIssues/LinkedData.html
[9] http://5stardata.info/images/5-star-steps.png

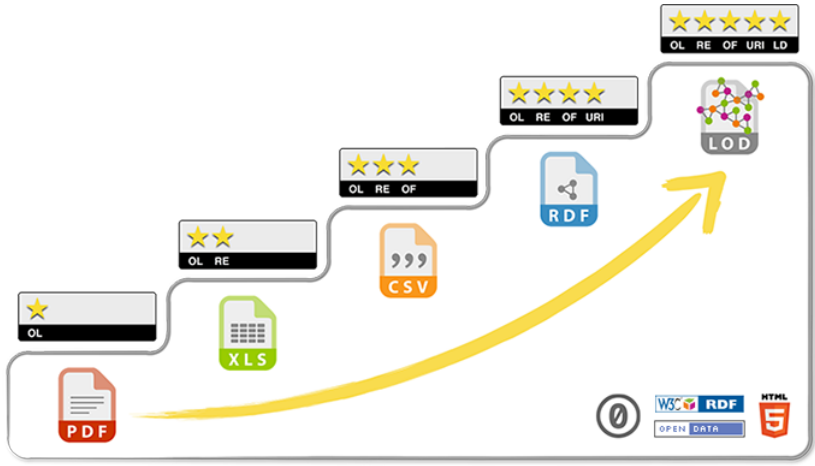


Figure 3.3: What is 5 star Linked Data? Cumulative scheme:

* Available on the Web with an open license, i.e. Open Data.

** Available as structured data (e.g., excel instead of an image).

*** Data in a non-proprietary format (e.g., CSV instead of excel).

**** Publish using open standards from W3C (RDF and SPARQL).

***** Link your data to other data to provide context.

datasets are DBpedia[10], which contains the structured information from Wikipedia's infoboxes, Geonames[11] is a geographical database with over 10 million geographical names,... More information on this project's time-line and accomplishments can be found online[12].

As of February 2018, the LOD cloud contains 2,973 datasets and a total of 149 billion statements. 99.9% of the triples in the LOD cloud are accessible via a queryable SPARQL interface [13].

To get an impression of the diversity of the Linked Data on the LOD cloud it's interesting to see the LOD visualizations[14] at different points in time. In 2009 we

[10] http://wiki.dbpedia.org/
[11] http://www.geonames.org/
[12] https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[13] http://stats.lod2.eu/
[14] lod-cloud.net

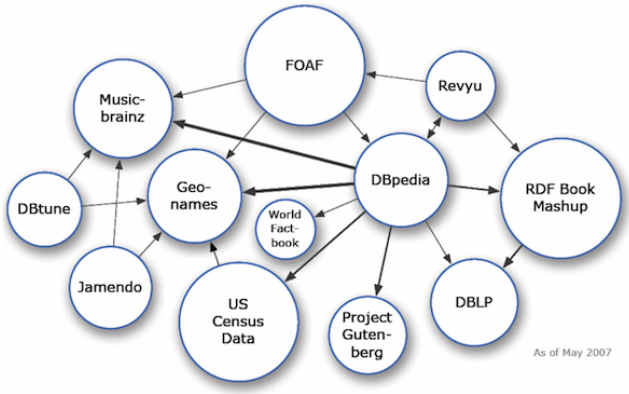


Figure 3.4: Linked Open Cloud Diagram in its initial shape in 2007.

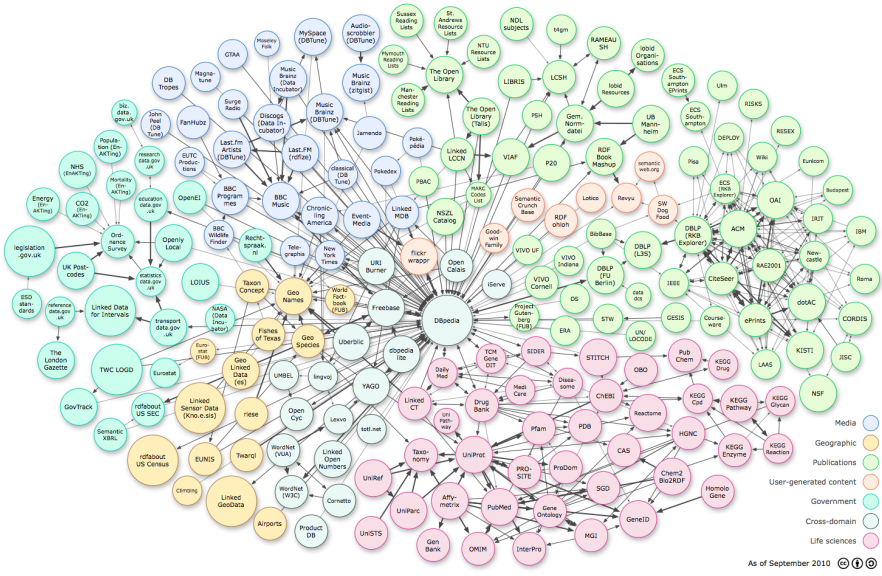


Figure 3.5: Linked Open Data Cloud as of September 2010. Already a significant and highly connected fraction of the cloud is taken up by Life Sciences datasets.

already see serious presence of Life Sciences data in the LOD cloud. Figure 3.5 shows some of the core datasets such as PubMed (medical publications), Drugbank (drugs and their targets), and UniProt (protein sequences). We will discuss these in more depth in chapter 7 when we describe Ontoforce's DISQOVER product, which is a user interface on top of the Life Sciences LOD cloud.

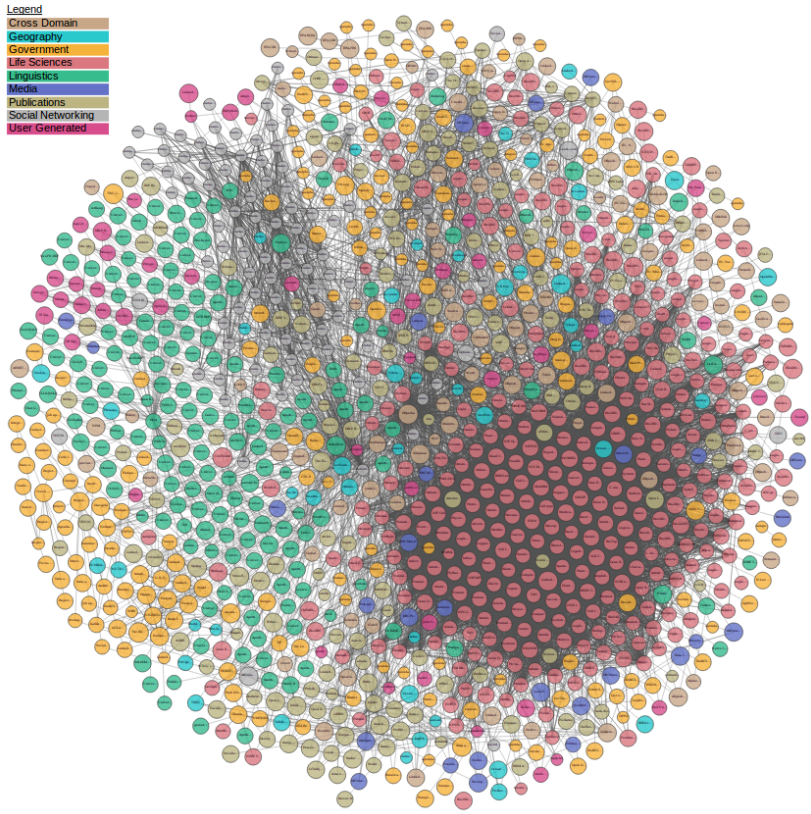


Figure 3.6: Linked Open Data Cloud as of June 2018. The success of the Life Sciences effort in the LOD cloud is hard to ignore. Especially the high degree of interconnections is remarkable.

The most striking is however the difference in composition in 2018, in Figure 3.6. Life Science datasets are by far the largest in number, but also in terms of their sizes. UniProt contains over 21 billion statements, Drugbank has 557 million and ChEMBL (properties of small molecules) has 445 million.

Given this evolution one could state that the Life Sciences domain is one of the most promising domains for Linked Data applications to be successful.

FAIR data Very similar to the LOD movement, recently (2016) a new movement within the Life Sciences domain was born. The followers agree to publish data according to the FAIR principles [34]: Findable, Accessible, Interoperable, and Reusable. One possible implementation is that of five-star Linked Data but FAIR data pays more attention to the aspect of privacy which is very important in the Life Sciences domain.

The rules in more detail (from the paper) are:

- **Findable:** This corresponds to having metadata with a globally unique and persistent identifier. The metadata has to be in a searchable resource.
- **Accessible:** Metadata is accessible via a standardized communication protocol which is open and free. The protocol supports security. Metadata are always accessible even when data is not.
- **Interoperable:** Metadata uses vocabularies and includes references to other metadata.
- **Reusable:** Metadata have rich and relevant attributes, has a clear usage license, has detailed provenance and is in agreement with domain-relevant community standards.

Semantic Web Technology Stack

So far we have described the ideas underlying Linked Data, but have not given any attention to their implementation. The Semantic Web Technology stack is shown in Figure 3.7 and shows the set of standards proposed by W3C[15].

- **RDF:** The Resource Description Framework [10] is the *syntax* of the Semantic Web. An RDF dataset is a directed labeled graph consisting of a number of triples. Each triple corresponds to an edge in this graph and represents a fact, consisting of a subject, a predicate, and an object. Everything (apart from literals) is described by URIs.
- **SPARQL:** The SPARQL Protocol and RDF Query Language [22] provides the means to consume Linked Data. The rules governing the Web API are called

[15] https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)

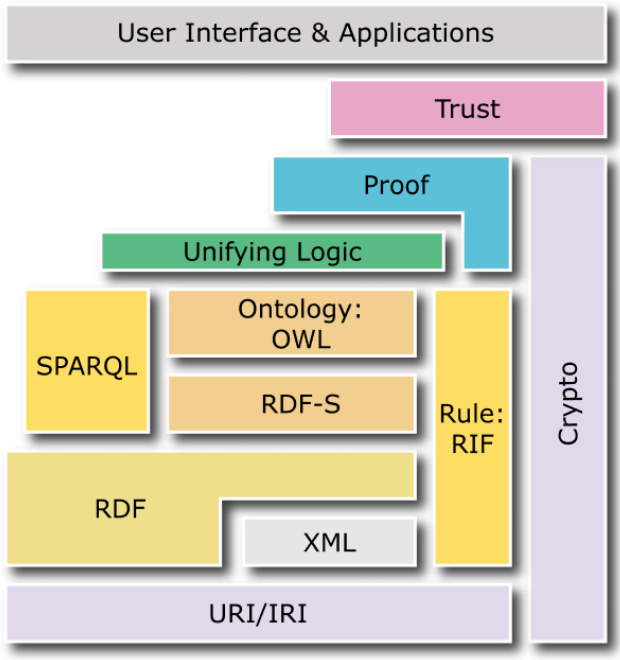


Figure 3.7: Semantic Web Technology stack as proposed by W3C. On the Semantic Web everything is expressed in terms of URIs. RDF is the syntax to express Linked Data. SPARQL is both a language and a protocol to query Linked Data. OWL and RDF-S are modeling languages. The other components are related to reasoning and are not within the scope of this thesis.

the SPARQL protocol and the query language is SQL-like language to query RDF graphs.

- **RDF-S** & **OWL:** RDF-Schema [7] is a language that is used to define Linked Data vocabularies/ontologies. OWL, the Web Ontology Language [20], has a much richer syntax to describe vocabularies.

Not all components of the Semantic Web stack have been realized yet. Especially interesting for the Life Sciences domain would be the realization of the *Crypto layer* to deal with the sensitive issue of patient data privacy. There is however support for 'transfer-crypto' as Semantic Web technologies now support the HTTPS. Old HTTP URIs can be transformed to HTTPS by means of redirection. Other security measures are the use of digital signatures and access control. The Semantic Web approach to access control can be found on the W3C page titled WebAccessCon-

trol[16]. The fact that there is no broadly accepted standard however does not prevent individual RDF database management systems to implement their own solution, such as the concept of graph-level security by Virtuoso[17]. We will revisit privacy issues in the Future Work chapter 9.

Let's explore the 3 main building blocks in more detail.

RDF

RDF[18] was already a W3C recommendation in 1999, prior to the emergence of the Semantic Web. It was initially conceived as an operating-system independent and vendor-neutral model for the exchange of metadata[19].

RDF is a graph-based model, with the graph being represented as a set of *triples*. Every triple corresponds to an edge in the RDF graph. An example of an RDF triple consisting of a <subject>, <predicate>, and an <object> is:

```
<http://dbpedia.org/resource/Noam_Chomsky>
      <http://dbpedia.org/ontology/birthPlace>
            <http://dbpedia.org/resource/Pennsylvania> .
```

This RDF triple states that Noam Chomsky is born in Pennsylvania. It is serialized using the *N-triples* format [8]. The latter is the most straightforward line-based serialization of RDF, with typically one triple per line and the dot as a line terminator.

The triple subject and predicate are always represented using URIs, but the object can also be a literal, such as a number or a string. In case the literal is a string optionally the language can be specified, other data types can also be specified. These cases are demonstrated below in an example dataset with triples linked to Noam Chomsky and the opening quote of this chapter:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .

dbr:Noam_Chomsky dbo:birthPlace dbr:Pennsylvania ;
           dbo:birthDate "1928-12-07"^^xsd:date ;
           dbo:field dbr:Linguistics , dbr:Political_criticism .
```

[16] https://www.w3.org/wiki/WebAccessControl
[17] http://docs.openlinksw.com/virtuoso/rdfgraphsecuritylevelrow/
[18] https://www.w3.org/2001/sw/wiki/RDF
[19] https://en.wikipedia.org/wiki/Resource_Description_Framework#History

```
dbr:Syntactic_Structures dbo:author dbr:Noam_Chomsky ;
          dbo:numberOfPages "117"^^xsd:integer ;
          dbo:Abstract "...offering the now-famous sentence
          'Colorless green ideas sleep furiously'..."@en .
```

The above dataset is serialized using *RDF-Turtle* syntax [3]. It supports the use of URI prefixes to make the URIs less verbose. If multiple triples share a subject or in case they share both subject and predicate these can be omitted using the ';' and ',' as line terminators. It is not our ambition to cover all the rules of the RDF serializations. One thing to pay closer attention to though, is the serialization of blank nodes, i.e., anonymous nodes with no URI specified[20].

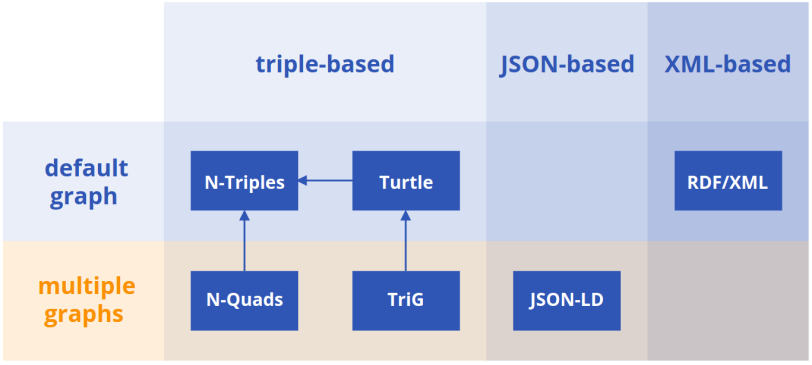


Figure 3.8: Schematic overview of the different RDF serialization types. A distinction is made between single (default) graph serializations and serialization with one or more named graphs.

Other RDF serialization types are RDF/XML [2] and JSON-LD [9]. An overview of all formats is given in Figure 3 (taken from course Web Fundamentals[21]).

All RDF-triples so far belong to one single unnamed graph. An RDF dataset can however also group triples in named graphs. The serialization format mentioned so far do not specify a graph name. To support this feature extensions are made available, such as N-Quads (superset of N-Triples), TriG (superset of Turtle)

Finally there is also an RDF syntax for embedding RDF directly into HTML tags: RDFa. A very popular ontology often embedded as RDFa in html web pages is the `schema.org` ontology maintained by Google.

[20] https://en.wikipedia.org/wiki/Blank_node#Anonymous_resources_in_RDF
[21] http://rubenverborgh.github.io/WebFundamentals/semantic-web/#rdf-syntaxes-schema

SPARQL

SPARQL is both a query language and a protocol for accessing a Web API using the SPARQL query language. As an example, let's have a look at the DBpedia SPARQL endpoint:

```
https://dbpedia.org/sparql
```

If we access this URI via a web browser, we can type our SPARQL queries directly in the Virtuoso User interface (Virtuoso is an RDF store which supports the SPARQL protocol). Contrary to the classical Web APIs which allow the consumer to only perform a limited set of actions, a SPARQL endpoint gives the end-user full access to the data. Full access in this case means that the consumer can send any SPARQL query to this interface.

As an example let's try to retrieve all information related to the resource Noam Chomsky:

```
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT ?p ?o
WHERE {
            dbr:Noam_Chomsky ?p ?o .
}
```

Note that the SPARQL query syntax bares a lot of similarity to the RDF Turtle syntax. The @-symbol has been dropped. If we want to see the SPARQL protocol in action we can for example open the web console in our browser. There we see that under the hood our browser performed an HTTP call use the GET verb of the following form:

```
GET https://dbpedia.org/sparql?query={...}
```

In between the curly brackets we have an URL-encoded[22] version of the SPARQL query string (to deal with unsupported characters such as spaces, newline in the string). Most SPARQL endpoints only support read-only operations using HTTP GET or POST. More information on the SPARQL protocol can be found in the specification [17]. In section 3.3, we'll have a closer look at the SPARQL query language.

[22] https://en.wikipedia.org/wiki/Percent-encoding

RDF-S & OWL

Describing 'data objects' can be done in a similar fashion, as is the case for Object-Oriented programming. RDF-S is a syntax for describing classes, properties, and data types or more generally *RDF ontologies*. These concepts are used to define certain constraints on RDF triples. For example, if we dereference `dbo:birthDate`, we get a number of triples describing the constraints of this predicate, some of these use the RDF-S:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <https://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/> .
dbo:birthDate rdf:type rdf:Property ;
           rdfs:domain dbo:Person ;
           rdfs:range xsd:date ;
           rdfs:label "birth date"@en .
```

The above triples assign a human-readable description to the predicate with `rdfs:label`, add a constraint to the subject (person) and the object (date), `rdf:type` assigns a class to the subject. OWL is a more powerful modeling language in the sense that more complex relations can be expressed. For example OWL allows the definition of transitive, symmetric, and inverse properties. We provide a toy example for clarification[23] below:

```
f:hasAncestor rdf:type owl:TranstiveProperty .
f:hasSpouse rdf:type owl:SymmetricProperty .
f:parent owl:inverseOf f:child .
```

The use of ontologies gives *reasoners* the ability to infer implicit knowledge. Say our RDF dataset contains only the following triple:

```
dbr:Noam_Chomsky f:hasSpouse dbr:Carol_Chomsky
```

Now, a reasoner armed with an RDF vocabulary describing the `hasSpouse` property, can infer a new fact:

```
dbr:Carol_Chomsky f:hasSpouse dbr:Noam_Chomsky
```

Reasoning agents come in two flavours: OWL reasoners and rule-based reasoners. OWL-reasoners infer additional facts by taking in a full ontology, rule-based reasoners allow more fine-grained control by specifying a specific set of rules, which

[23] `https://www.w3.org/2007/OWL/wiki/PrimerExampleTurtle`

can be used for inferring additional facts. More insights into the performance of the different reasoning agents can be found in benchmark papers of OWL reasoners [21] and rule-based reasoners [24].

3.2 Publishing Linked Data

The previous sections made the case for putting Life Sciences data on the Web as Linked Data. However, there still lies a lot of work ahead, which is clearly decribed using a quote of the Course Web Fundamentals[24]: "Most Linked Data isn't born linked. Most Linked Data isn't born at all."

Linked Data requires an ETL[25] step, in which the original data is transformed into a Linked Dataset. We will cover this in the subsection on mapping Linked Data. When this *mapping* step is completed the data can be put on the Web as-is, but there exist alternatives which will be discussed in the subsection on putting Linked Data on the Web.

Mapping Linked Data

The Life Sciences data consumed and produced by the BLSSpeller algorithm is *structured* data, but not Linked Data! The conversion of non-RDF to RDF data is called a *mapping* process. The RDF Mapping Language (RML), developed by Dimou et al. [15], enables the expression of 'mapping rules' to convert any type of (structured) data to an RDF serialization. These mapping rules describe how the URIs can be generated from the original data sources, typically by defining URI templates. This describes the physical mapping process, executed by the RML processor[26], but this does only lead to 4* Linked Data. In order to create 5* Linked Data, also links crossing the boundaries of the single dataset are required, which effectively make the connection with the LOD Cloud.

The mapping rules describe which triples can be generated from the input data. An important step, ensuring the usability of the Semantic Web, is to maximally reuse the existing vocabularies to define properties and classes. The Open Knowledge Foundation currently hosts the Linked Open Vocabularies project (LOV), which provides an overview of existing ontologies (590 as of February 2018) and tools, such as

[24] http://rubenverborgh.github.io/WebFundamentals/linked-data-publishing/#origins
[25] `https://en.wikipedia.org/wiki/Extract,_transform,_load`
[26] http://rml.io/RMLsoftware.html

a vocabulary search engine, to enable the retrieval of useful vocabularies to support a certain project[27].

For an in-depth coverage of RML we refer to the PhD thesis of Dr. Dimou [14]. W3C also published a set of *best practices* for publishing Linked Data[28].

To deal with unstructured data such as natural language other techniques exist, for example DBpedia Spotlight[29] which, among others, makes use of Named Entity Recognition.

Putting Linked Data on the Web

Once the mapping phase is completed, all that is left is providing an interface to allow the data to be consumed by a third party. In the context of the Semantic Web, the following 3 approaches are the most common:

- Publication as a **Data Dump**: The RDF dataset is serialized in a number of files which can be downloaded. Further consumption of the data is the sole responsibility of the end-user. The advantage here is mainly the lack of effort on the side of the data publisher. This is obviously a trade-off and therefore a drawback for the data consumer, especially when the dataset is big and/or regularly updated. The entire LOD cloud published as RDF data dumps[4] can be found on the LOD Laundromat website[30].

- Publication as a set of connected **Linked Data Documents**: The data related to Noam Chomsky has so far been explored by making use of Linked Data Documents. Each of these pages provides an overview of both the incoming and outgoing links related to the resource it represents. Approaches exist to execute queries on top of Linked Data documents [23], but the need to dereference the individual URIs makes this a less efficient approach compared to a SPARQL endpoint.

- Publication as a **SPARQL endpoint**: Publication making use of an RDF database management system and by making use of a Web API supporting the SPARQL protocol. This publishing scheme is the most flexible for the end-user, who can

[27] `http://lov.okfn.org/dataset/lov/`
[28] `https://www.w3.org/TR/ld-bp/`
[29] `http://www.dbpedia-spotlight.org/`
[30] `http://lodlaundromat.org/`

formulate arbitrary queries on the data, but requires the most effort from the data publisher who now also has to guarantee the availability of the endpoint. The latter is both expensive and hard to fulfill and one of the open questions in the Semantic Web community.

3.3 Consuming Linked Data: SPARQL and RDF Databases

A database management system typically supports four categories of operations, summarized in the acronym CRUD: Create, Read, Update and Delete[31].

These operations are expressed using a *query language*. In relational database management systems this is the Structured Query Language (SQL[32]). RDF is a graph-based data model and comes with its its own query language: SPARQL. SPARQL is a standardized graph query language [22] and reuses many of the ideas of SQL[33] while avoiding 2 of its restrictions:

1. SPARQL can be used to perform (federated) queries which target **multiple databases at once**, these databases require only one thing in common: they have to support SPARQL;

2. SPARQL doesn't require data to be organized in tables, but is used for pattern matching in **schemaless** graphs.

Another important difference with relational database management systems, is that RDF data can also be used to infer additional knowledge. A *reasoning agent* typically uses an ontology or a set of rules to infer additional RDF triples, as was previously described in the section RDF-S & OWL on page 41. The combination is possible, in which case SPARQL queries are run with support for an *entailment* regime (for example OWL or RDF-S).

We will discuss the basics of SPARQL in the next section, the subsequent section we will devote to RDF database management systems. These systems are used in the back-end of SPARQL Web APIs, to process the incoming queries.

[31] `https://en.wikipedia.org/wiki/Create,_read,_update_and_delete`
[32] `https://en.wikipedia.org/wiki/SQL`
[33] The typical SELECT <> FROM <dataset> WHERE <pattern>

SPARQL: a semantic graph query language

An in-depth coverage of the SPARQL specification is not within the scope of the thesis. Instead, we will cover SPARQL in an example-driven way, such that the reader can follow along in chapter 7 when we study the effect of certain SPARQL query features on the different system run times

The most elementary SPARQL query is a basic graph pattern (BGP). In the next query we try to retrieve the spouse of Noam Chomsky, when she was born, and when she passed away:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?spouse, ?name, ?born, ?died
FROM <http://dbpedia.org>
WHERE {
           dbr:Noam_Chomsky dbo:spouse ?spouse .
           ?spouse rdfs:label ?name .
           ?spouse dbo:birthDate ?born .
           ?spouse dbo:deathDate ?died .
} LIMIT 1
```

The result of this query is a table (with a single row), containing the variable bindings:

```
("http://dbpedia.org/resource/Carol_Chomsky",
"Carol Chomsky"@de,
1930-07-01,
2008-12-19)
```

As explained in the course materials[34] provided by the EUCLID project[31]: a SPARQL query consists of a number of components:

1. A **Prologue** contains the prefix definitions, such as `dbr`, `dbo`, and `rdfs`.
2. A **Query Form**: The SELECT operator returns a table with the *bindings* to the variables in the BGP. For other query forms we refer to the specification [22]: ASK, DESCRIBE, and CONSTRUCT.
3. An (optional) **Dataset Specification** using the FROM clause. Omitting the FROM statement matches the BGP against the *default graph* which is the union of all triples, irrespective of the graph to which they belong.

[34] `http://euclid-project.eu/modules/course2.html`

4. A **Query Pattern**: a basic graph pattern described using the Turtle syntax contain a series of variables (a variable is written with a question mark, for example: `?x`). In the previous example the pattern consists of 4 edges all connected with the variable `?spouse`.

5. An (optional) **Solution Modifier** with the ability to reorganize the table output. Examples are ORDER BY, GROUP BY, FILTER, DISTINCT,... In the example LIMIT is used, since Carol Chomsky has different spellings per language, which we want to omit.

One striking difference with SQL is the fact that *joins* in the SPARQL language are implicit! The BGP pattern in fact corresponds to 3 inner joins of the tables resulting from the individual triple queries.

The equivalent of a left outer join comes with the use of the keyword OPTIONAL. We could for example modify the previous query to extract the birth and death date of Noam Chomsky, keeping into account that the latter might be missing (for obvious reasons) (prefixes omitted):

```
SELECT ?born, ?died
WHERE {
            dbr:Noam_Chomsky dbo:birthDate ?born .
            OPTIONAL { dbr:Noam_Chomsky dbo:deathDate ?died . }
} LIMIT 1
```

with no binding for the `?died` variable. Note that without the OPTIONAL operator the query would have no matches!

The most common modifier in the benchmarks in chapter 7 is the FILTER operator. The example retrieves the URI of the book containing this chapter's quote ('Syntactic Structures'):

```
SELECT ?book
WHERE {
            ?book dbo:abstract ?abstract
            ?book rdf:type dbo:Book
            FILTER ( regex(str(?abstract), "Colorless green ideas") )
}
```

The original SPARQL 1.0 specification [26] was mainly focused on simple BGP query matching and is what we have covered so far. The new features provided by the

SPARQL 1.1 specification[35] fall into 3 categories: (i) query extensions, (ii) support for updates, and (iii) support for query federation.

The query extensions consist of the support for aggregations such as COUNT, negations such as FILTER NOT EXISTS { TRIPLE }, sub-queries such as a nested SELECT, property paths, ...

Property paths are used for *path queries*, where a 'chain' of predicates can be matched. An example of a property path query could be, to retrieve all ancestors of Noam Chomsky via the * operator:

```
SELECT ?name
WHERE {
            ?ancestor dbo:child*  dbr:Noam_Chomsky .
            ?ancestor rdfs:label ?name .
}
```

As an example of an aggregate operator, we could count the number of distinct co-authors of his manuscripts:

```
SELECT COUNT DISTINCT ?coauthor
WHERE {
            ?manuscript dbo:author dbr:Noam_Chomsky .
            ?manuscript dbo:author ?coauthor
}
```

Updates are not relevant for this thesis and are not supported in public SPARQL endpoints so far.

SPARQL 1.1 provides support for federated querying via the SERVICE operator, with which a specific SPARQL endpoint can be targeted with a (sub-)query. To indicate that a certain BGP in a WHERE clause needs to be resolved by a specific SPARQL endpoint, this set of triples can be enclosed as shown in the following example:

```
SERVICE <http://dbpedia.org/sparql> {
            dbr:Noam_Chomsky ?p ?o
}
```

For an instructive overview of all SPARQL features we refer to the SPARQL cheat sheet[36].

[35] https://www.w3.org/TR/sparql-features/
[36] http://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf

RDF Databases

An RDF database management system is responsible for handling SPARQL queries. In a first step a query needs to be parsed, with the help of a SPARQL algebra[37]. This results in a logical query plan, which can be (optionally) further optimized. Finally, this query plan is executed against the RDF dataset.

In a naive approach RDF can be stored in a three-column SQL table. This however ignores many of the specifics of RDF and SPARQL and has been shown to be an inefficient approach to resolving SPARQL queries. Unique to RDF storage is, for example the abundance of URIs, which can be quite verbose and repetitive. This can be exploited by appropriate string encoding. There exists a plethora of systems for RDF database management. These can be classified along multiple dimensions [11]:

1. Different approaches to **encoding URIs** into dictionaries;
2. Different **storage layouts**: Both native and non-native, i.e., the data is stored in a different type of database management systems, for example a relational database [16];
3. Different approaches to **indexing** in RDF databases;
4. Different approaches to query plan **optimizations**;
5. Different **partitioning** schemes for dealing with Big Linked Data;
6. Different approaches to **federated querying** over multiple databases;
7. Different support for **reasoning** services.

Chapter 5 of the book 'RDF Database Systems' [11] provides an in-depth survey of storage layouts and indexing schemes for RDF databases.

3.4 Big Linked Data

In the introduction of this chapter we mentioned that the goal of the Semantic Web is to serve as a *global distributed database*. The LOD cloud is a realization of this goal. Building an application on top of even a subset of the LOD cloud is still challenging,

[37] `https://www.w3.org/TR/sparql11-query/#sparqlAlgebra`

as this amount of data qualifies as Big Data, with the most important 'V' this time being the *Volume*.

Dealing with increasing volumes of Linked Data can be addressed by:

- **Vertical scaling** is using an expensive high-end machine with a lot of RAM and CPU power. No additional software development is required in this case.
- A **Compression algorithm** such as HDT [18] can easily compress RDF datasets by a factor of 10–20. This allows for much larger datasets to be handled by a single server.

Each of the previous options comes at a price. For vertical scaling this is literally that memory is limited and expensive. Compressing the RDF data, increases the query latency significantly and severely limits the number of systems, which can be used to host the data. Opting for a *distributed architecture* might therefore be a serious consideration, with three major options to choose from:

- **Horizontal scaling** uses multiple - often cheap, low-end - instances in *homogeneous* a distributed system. Most enterprise RDF stores support parallelization, but this can imply both a high availability solution (data replication), or a sharded system (data partitions) that can deal with increasingly large datasets.
- **Query federation** [30]: All datasets are hosted by their providers and a federated query engine redirects the relevant parts of each query to the right endpoint and finally combines all the received information to solve the query. In this context the full distributed system can be *heterogeneous*.
- Native **Big Data approaches** typically map SPARQL queries to SQL technologies available in the Hadoop stack. These approaches are currently still in the proof-of-concept phase, examples are SparkSQL [12] or Impala [29].

Other types of interactions with graph data The topic of (RDF) graph querying can be easily confused with the similar concepts of graph algorithms and that of (global) graph processing. The systems discussed here (and in chapter 7) are graph databases which support the SPARQL protocol. Other graph databases such as Neo4J, have a different query language. However, Neo4J does have an unofficial

SPARQL plugin[38], it is currently no longer maintained and does not support the full SPARQL language.

The Neo4J database is, due to its internal representation relying on Linked Lists[39], specialized in *graph algorithms*. These algorithms often rely on 'traversals', where one has to efficiently navigate through the graph, for example to find the shortest path between two nodes.

The primary goal of RDF database systems however is graph *pattern matching*, which we discussed in the context of BGPs.

Finally there are also *global graph processing* operations. For example calculating the PageRank of all nodes in a graph. Typical systems to perform these kind of operations in parallel are Apache Giraph, Apache Hama, or Spark's GraphX system, all of which are based on Google's Pregel [25] project. These systems have a different model for parallelization called Bulk Synchronous Processing.

Homogeneous Systems

In homogeneous distributed systems all nodes are basically identical, running the same software. This way the system behaves very similarly to a centralized database. An example of such a system is Open Link's Virtuoso Cluster edition[40]. Systems in this category usually have a shared-nothing architecture (master-slave approach), where all interactions go via the master-node and the slave nodes do not interact with each other.

An important factor in the performance in this type of systems is how the data is partitioned between the slave nodes. Typically a *horizontal partitioning* scheme is used, where every slave node gets a subset of the total RDF dataset. To determine these subsets different allocation strategies exist [11]:

- **Random allocation:** Triples are randomly distributed, which leads to inefficient queries since there is no optimization in terms of resource usage.
- **Hash-based:** Typically a hashing function is used on the triple subject. This way all triples with the same subject end up on the same slave node. Star

[38] https://github.com/neo4j-contrib/sparql-plugin
[39] https://stackoverflow.com/questions/24366078/how-neo4j-stores-data-internally
[40] More info: http://docs.openlinksw.com/virtuoso/clusteroperation/

queries, with a common subject, therefore only need to be sent to a single slave.

- **Range-based:** Every node gets a range of URIs assigned, this way URIs with a common prefix are stored in the same slave node. This might lead to an unbalanced distribution however.
- **Graph-partitioning:** The idea is here to increase the change of local graph patterns to match fully in a single node. All nodes which are within a certain number of hops should end up on the same slave.
- **Query-load based:** This approach is driven by the consumption behavior. Resources that are accessed together, given a query, are more likely to be stored on the same slave.

It might also be beneficial to use a distributed system, even when the queries can still be handled by a centralized approach. The goal in this case is to secure the *High Availability* of the system. High availability[41] corresponds to a certain service-level agreement, in terms of system uptime. In this case, data can be *replicated*, such that every slave node has a full copy of the data. The master then serves as a load balancer. In a live setting this will decrease the server load, since less queries have to be handled within the same time span. This can lead to a small speedup for the individual queries, but the main effect is the query throughput: a batch of N queries can be processed with a linear speedup. In horizontal scaling, the primary goal is however to support *scaling out*, i.e., serving bigger datasets with more resources but ideally no decrease in performance.

Heterogeneous Systems with Federation

In the Semantic Web Linked Data publishers and consumers are decoupled by the SPARQL interface. Thus, collecting and ingesting a set of Linked Data dumps in a centralized homogeneous system is not in line with the Semantic Web vision. Querying the data as-is, with no restrictions on the back-end systems (apart from the SPARQL Web API), requires a new approach to query processing: *Federated Querying*. Federated querying circumvents the ETL step needed in homogeneous systems: each time a data publisher updates the data in an endpoint, the centralized system is lagging and has to re-run the ETL pipeline.

[41] `https://en.wikipedia.org/wiki/High_availability`

A federated query engine acts as a mediator: the end-user sends a SPARQL query to this system and the mediator redirects the relevant parts of the query to the appropriate SPARQL endpoint and integrates the results of the different sub-queries. This however, comes with its own set of new challenges:

- **Source selection:** A query federation system must be able to select the appropriate endpoints to resolve a query and avoid that irrelevant queries are being sent to SPARQL endpoints.
- **Query Plans:** The mediator must ideally run joins locally in an endpoint whenever possible. Some systems make use of statistics to help select the optimal order in which sub-queries are executed.
- **Data Transfer:** Data transfer from sources to mediator should be kept at a minimum to reduce the latency but also to preserve the resources of the mediator (memory).

BigRDFBench [28] is a benchmark comparing the most relevant systems on all these dimensions. The paper discusses FedX [30], SPLENDID [19], ANAPSID [1], and combinations.

In Federated querying it is not possible to control the partitioning of the data, as this is controlled by the data publisher, which takes away an important tool for optimization.

In the previous section on homogeneous systems we discussed the aspect of High Availability. For federated systems this is challenging requirement, as the individual endpoints are not managed by the mediator. High availability turns out to be a real issue for the Semantic Web, already for single SPARQL endpoints. The main cause of this availability issue is the flexibility of the SPARQL Web API: an end-user can formulate an arbitrary complex query. This makes it impossible to guarantee, that the web server will have sufficient resources, to respond to all incoming queries. This issue has to be addressed, in order to make a federated querying a viable alternative to the centralized approaches.

Verborgh et al [33] proposed a solution for this availability issue by introducing the concept of *Linked Data Fragments*. As can be seen on Figure 3.9 (taken from Web Fundamentals course[42]), different approaches for publishing Linked Data can be

[42] `http://rubenverborgh.github.io/WebFundamentals/linked-data-publishing/#triple-pattern-fragments`

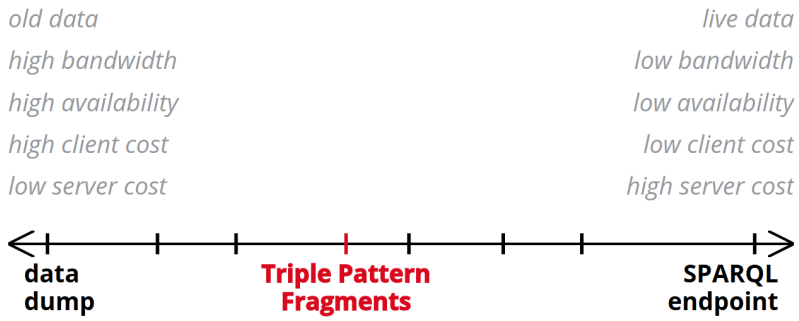


Figure 3.9: Linked Data Interfaces on an axis each with their trade-offs. Triple Pattern Fragments offers: live data, medium bandwidth, high availability, medium client cost, low server cost.

put on an axis, with the data dump and the SPARQL endpoint being the extremes. In this work the Triple Patterns Fragments interface is proposed. In this setup the server's API is limited, such that it only has to answer (single) triple pattern queries, which greatly benefits the availability and requirements of this system. Using this approach SPARQL queries are solved as a coordinated effort between client and server. The client directs and integrates the results of the triple patterns queries. Note that this is very similar to the concept of federated querying, and in fact the TPF implementation supports federated querying out of the box! This new type of interface is still a research-in-progress, but its status will be assessed in chapter 7 on RDF benchmarking. More information on Linked Data Fragments research and live demos can be found on the project website[43].

3.5 Conclusion

The Life Sciences domain could greatly benefit from publishing its data as Linked Data. Both the availability of many Linked Datasets in this domain and the fact that Linked Data addresses the requirements for a successful publishing strategy. Linked Data is self-descriptive, preventing a loss of semantics after publishing support this claim. Furthermore, data can be consumed without requiring an interaction with the data publisher. Linked Data published according to the principles of the five-star scheme results in connected datasets which are automatically integrated.

Once Linked Data has been generated, it has to be published. This chapter high-

[43] http://linkeddatafragments.org/

lighted several approaches for data publishing and explained the prior step of Linked Data generation. The requirement for a queryable web interface, can be addressed by turning to SPARQL endpoints. Ongoing research is also looking into interfaces with a higher availability such as the Triple Pattern Fragments interface.

Finally, the section on Big Linked Data also demonstrates, how Big Data should be handled in the Semantic Web context, both for single Big Datasets and for physically distributed datasets.

Acknowledgement

This main inspiration for this chapter came from the course Web Fundamentals[44] taught by Prof. Verborgh, the PhD dissertations of Dr. De Vocht [13], Dr. Vander Sande [32] and Dr. Dimou [14], as well as the book RDF Database Systems [11]

[44] https://rubenverborgh.github.io/WebFundamentals/

References

[1] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. *International Semantic Web Conference*. Springer, pages 18–34, 2011.

[2] D. Beckett and B. McBride. Rdf/xml syntax specification (revised). *W3C recommendation*, 10(2.3), 2004.

[3] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. Rdf 1.1 turtle. *World Wide Web Consortium*, 2014.

[4] W. Beek, L. Rietveld, H. R. Bazoobandi, J. Wielemaker, and S. Schlobach. Lod laundromat: a uniform way of publishing other people's dirty data. *International Semantic Web Conference*. Springer, pages 213–228, 2014.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[6] C. Bizer, T. Heath, D. Ayers, and Y. Raimond. Interlinking open data on the web. *Demonstrations track, 4th european semantic web conference, innsbruck, austria*, 2007.

[7] D. Brickley. Rdf vocabulary description language 1.0: rdf schema. *http://www. w3. org/TR/rdf-schema/*, 2004.

[8] G. Carothers and A. Seaborne. Rdf 1.1 n-triples: a line-based syntax for an rdf graph. *World Wide Web Consortium. http://www. w3. org/TR/n-triples/. Accessed*, 24, 2014.

[9] W. W. W. Consortium et al. Json-ld 1.0: a json-based serialization for linked data, 2014.

[10] W. W. W. Consortium et al. Rdf 1.1 concepts and abstract syntax, 2014.

[11] O. Curé and G. Blin. RDF database systems: triples storage and SPARQL query processing. Morgan Kaufmann, 2014.

[12] O. Curé, H. Naacke, M. A. Baazizi, and B. Amann. On the evaluation of RDF distribution algorithms implemented over apache spark, 2015.

[13] L. De Vocht. Exploring semantic relationships in the web of data. PhD thesis. Ghent University, 2017.

[14] A. Dimou. High quality linked data generation from heterogeneous data. PhD thesis. University of Antwerp, 2017.

[15] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. Rml: a generic language for integrated rdf mappings of heterogeneous data. *LDOW*, 2014.

[16] D. C. Faye, O. Cure, and G. Blin. A survey of rdf storage approaches. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 15:11–35, 2012.

[17] L. Feigenbaum, G. T. Williams, K. G. Clark, and E. Torres. Sparql 1.1 protocol. *Recommendation, W3C, March*, 2013.

[18] J. D. Fernández, M. A. MartıÍnez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 19:22–41, 2013.

[19] O. Görlitz and S. Staab. Splendid: SPARQL endpoint federation exploiting void descriptions. *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*. CEUR-WS. org, pages 13–24, 2011.

[20] W. O. W. Group et al. Owl 2 web ontology language document overview, 2009.

[21] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[22] S. Harris, A. Seaborne, and E. Prud'hommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10), 2013.

[23] O. Hartig. An overview on execution strategies for linked data queries. *Datenbank-Spektrum*, 13(2): 89–99, 2013.

[24] S. Liang, P. Fodor, H. Wan, and M. Kifer. Openrulebench: an analysis of the performance of rule engines. *Proceedings of the 18th international conference on World wide web*. ACM, pages 601–610, 2009.

[25] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, pages 135–146, 2010.

[26] E. Prud, A. Seaborne, et al. Sparql query language for rdf, 2006.

[27] R. Reiter. On closed world data bases. In: *Readings in artificial intelligence*, pages 119–140. Elsevier, 1981.

[28] M. Saleem, A. Hasnain, and A.-C. N. Ngomo. Bigrdfbench: a billion triples benchmark for SPARQL endpoint federation.

[29] A. Schätzle, M. Przyjaciel-Zablocki, A. Neu, and G. Lausen. Sempala: Interactive SPARQL Query Processing on Hadoop. *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Proceedings, Part I*, pages 164–179, 2014.

[30] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. *The Semantic Web - ISWC 2011, Proceedings, Part I*, pages 601–616, 2011.

[31] E. Simperl, M. Acosta, M. Dimitrov, J. Domingue, P. Haase, M. Maleshkova, A. Mikroyannidis, B. Norton, and M. E. Vidal. Euclid: educational curriculum for the usage of linked data. *2012*, 2014.

[32] M. Vander Sande. Metadata and control features for low-cost linked data publishing infrastructures. PhD thesis. Ghent University, 2018.

[33] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37–38:184–206, March 2016.

[34] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

Part II

Distributed Pattern Mining using Comparative Genomics

Chapter 4

DNA Sequence Data

> *The essence of life is statistical improbability on a colossal scale.*
>
> —Richard Dawkins.

Studying the machinery which is common to all organisms in nature, such as DNA replication or gene transcription, is a humbling experience. The fact that these complex molecular processes are the result of evolution is almost impossible to grasp. One reason for this is that it goes against our intuition of the Second Law of Thermodynamics. This law states that in a closed system the total entropy can never go down. However, as our planet is not a closed system, this obviously does not lead to a contradiction.

The probability of generating an 'initial DNA molecule' by chance is extremely small. However, the only reasonable explanation to why it came to be might lie in the *weak anthropic principle*. This explains that there is a certain survivor bias to this observation: "only in a universe capable of eventually supporting life will there be living beings capable of observing and reflecting on the matter."[1]

In this chapter we will provide a biological primer. This will give the reader the basic insights needed to understand the biology behind the data for the datasets used in part II of this work.

In section 4.1 we will describe the historical context, eventually leading to the discovery of DNA. Section 4.2 introduces our common understanding of evolutionary

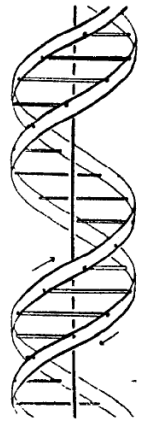
[1] Weak Anthropic Principle: `https://en.wikipedia.org/wiki/Anthropic_principle`

processes and the role of homology in the studying them. In section 4.3 we discuss the mechanism of gene transcription and regulatory elements.

4.1 The discovery of DNA

The discovery of the elementary building blocks of life has some parallels with the discovery of the elementary building blocks of nature. In 1665 Robert Hooke studied biological organisms using a microscope. He discovered that these organisms are composed of similar structures, which he coined *cells*. The Greek philosophers proposed the analogous idea of the atom, an indivisible unit of matter.

Just like the periodic table contains many elements, cells come in many different shapes. They do have many commonalities as well, most importantly their ability to self-replicate. This self-replication mechanism has been the subject of decades of research. The goal of this research was to find a chemical molecule, with the right properties to enable the long-term storage of genetic information.



This figure is purely diagrammatic. The two ribbons symbolize the two phosphate—sugar chains, and the horizontal rods the pairs of bases holding the chains together. The vertical line marks the fibre axis

We wish to put forward a radically different structure for the salt of deoxyribose nucleic acid. This structure has two helical chains each coiled round the same axis (see diagram). We have made the usual chemical assumptions, namely, that each chain consists of phosphate diester groups joining β-D-deoxyribofuranose residues with 3′,5′ linkages. The two chains (but not their bases) are related by a dyad perpendicular to the fibre axis. Both chains follow right-handed helices, but owing to the dyad the sequences of the atoms in the two chains run in opposite directions. Each chain loosely resembles Furberg's[2] model No. 1; that is, the bases are on the inside of the helix and the phosphates on the outside. The configuration of the sugar and the atoms near it is close to Furberg's 'standard configuration', the sugar being roughly perpendicular to the attached base. There

Figure 4.1: Screenshot of the 1953 breakthrough paper of Watson and Crick describing the structure of DNA.

Deoxyribonucleic acid (DNA) was discovered in the 19th century. It had been overlooked for a long time, as researchers believed it to be a simple repetitive molecule. While Mendel already predicted the existence of genes, the units of hereditary information, it was only in the 1940s and 1950s that multiple breakthroughs led to their

identification. X-ray images produced by Rosalind Franklin and Maurice Wilkins suggested that DNA had a helical structure. Watson and Crick managed to finally propose a structure for the DNA molecule which explains Chargaff's rule. This empirical rule states that the concentration c_{base} of the DNA bases is as follows: $c_A = c_T$ and $c_C = c_G$.

In their 1953 paper, describing the properties of the DNA molecule [13], Watson and Crick made an interesting postulate:

"It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material."

Figure 4.1 is a screenshot of their breakthrough paper, with an image of the proposed double helix structure given.

DNA $\mapsto$ RNA $\mapsto$ Proteins The genetic information is encoded in an alphabet consisting of 4 nucleotides (monomers): A (adenine), C (cytosine), G (guanine) and T (thymine). The two DNA strands are complementary, meaning that A and T as well as C and G are always bound together (Chargaff's Rule) and form the 'steps' of the spiral stairs.

DNA fulfills the role of a library which possesses the blueprints to create proteins. Proteins are macromolecules consisting of long chains of amino acids. They take the lead part in many biochemical reactions in living organisms[2]:

- They can function as enzymes, which catalyze chemical reactions.
- They are involved in cell signaling. The most prominent protein in this context is insulin.
- They can provide structural properties, such as the keratin protein which occurs in hairs and nails.

The amino acids make up for a 20-character alphabet. It has been shown that triplets of DNA (codons) map to amino acids. There is redundancy in this mapping as there are 4^3 codons.

The protein-coding regions in DNA are called *genes* and make up about 1% of a cell's entire genetic code or its *genome*.

[2] `https://en.wikipedia.org/wiki/Protein`

The process in which information is 'borrowed' from this gene library is common to all living organisms and is coined the *'Central Dogma of Molecular Biology'*: The DNA molecule is read and *transcribed* into an RNA (ribonucleic acid) molecule, which is much smaller and more reactive than the almost inert DNA molecule. Important is the RNA molecule's ability to exit the cell's nucleus where it can be *translated* into proteins by the ribosomes.

This section is based partially on the biology primer in Bioinformatics Algorithms by Pevzner [7].

4.2 Evolutionary Processes and the Role of Homology

Processes which lie at the basis of evolution are situated in both mitosis and meisosis. Typically these evolutionary processes are linked to *erratic behavior*, which can result in different types of mutations:

These mutations can manifest themselves at very different scales:

- **Point mutations** [5]**:** Base substitutions, insertions and deletions.
- **Gene and Chromosome mutations** [6]**:** Duplication, translocation, loss and inversion of genes and gene segments.
- **Genome level:** Whole genome duplications[3, 11, 12] changing the ploidy[3] level of an organism.

Duplications are very important to evolution, as they introduce redundancy and therefore 'fuel for innovation' [8]. These duplications can occur during DNA replication, for example during a process called *replication slippage*, but also during DNA recombination, for example as the result of *ectopic recombination*. More details on the gene duplication origins can be found on the wiki page[4].

Whole-Genome Duplications have been shown to have played a key role in the survival of mass extinction events [4]. WGDs are the result of erratic behavior in sexual reproduction, where an organisms receives two copies of the entire genome from each of its parents instead of one[5].

[3] Ploidy is the number of times each chromosome occurs in a cell's genome. Humans for example have a pair of each chromosome and are therefore diploid. Polyploidy is very common in plants.
[4] Mechanism for gene duplication: `https://en.wikipedia.org/wiki/Gene_duplication`
[5] Mechanism for Whole-Genome Duplication: `https://genomevolution.org/wiki/index.php/Whole_genome_duplication`

Homologous regions Reconstructing genome evolution relies on the identification of homologous regions both *within* a single genome as *between* genomes of related species. Homologous segments share a significant fraction of gene content. When only content is conserved, we refer to *syntenic* regions, whereas when also gene order is preserved we use *collinear* regions [10].

Detecting homologous regions, apart from studying genome evolution, enables the transfer of genome annotations from one organism to another and can help with the identification of noncoding sequence elements present in the vicinity of the homologous genes [9].

In chapter 5 we will discuss the inner workings of an algorithm, called i-ADHoRe, which is specifically designed to identify homologous regions.

4.3 Regulatory Elements

Central Dogma Genes carry all the information that is required to build proteins. The storage medium for this information is the DNA molecule. The 'reading' mechanism, which is shared between all living organisms, corresponds to a pathway which is labeled 'the Central Dogma of molecular biology'. This pathway is shown in Figure 4.2.

The central dogma does however not tell the entire story. In fact, if we consider Figure 4.2 as a directed graph with two edges going from DNA to proteins, it turns out that in reality some additional edges should be drawn. First of all, proteins can bind to the DNA sequence, thereby controlling the rate of gene expression. Secondly, not all RNA molecules are intended for protein assembly. The exact role of these RNAs is still the subject of active research. This becomes already apparent when comparing different editions of 'The Cell'. In the sixth edition (2015) a new type of RNA was added: long non-coding RNAs were not mentioned in the fifth edition of the book (2008).

The fact that this directed graph does not tell the entire store also demonstrates why it might not be possible to demystify the entire transcription machinery if we focus our attention on the DNA sequence alone, which is the topic of chapter 6. The penultimate solution will require taking into account the entire set of gene products in the graph.

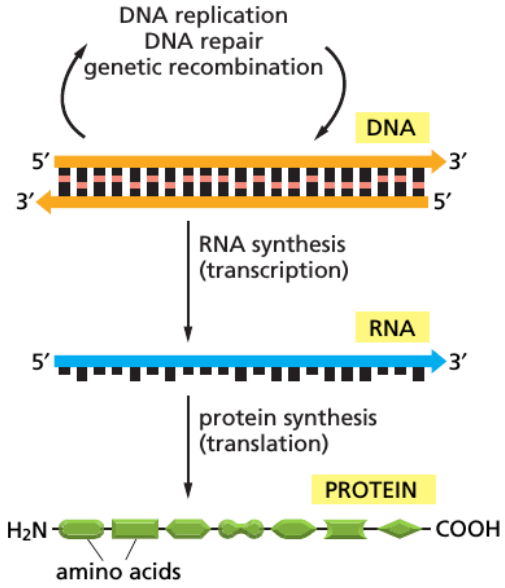


Figure 4.2: Central Dogma of molecular biology. Proteins (green) are created in a two-step process. First the DNA molecule is read and *transcribed* in a single-stranded RNA molecule which can leave the cell's nucleus. In the cytoplasm the RNA is converted into a protein in the ribosomes.

Transcriptional Regulation Knowing how information flows from DNA to proteins alone is not sufficient to understand the cell's ability to respond to a change in environment. This reactive behavior can be mainly attributed to the concept of transcriptional regulation: the rate at which a gene is transcribed (=gene expression) into RNA can change.

The process of transcribing DNA into RNA has many similarities with DNA replication. *RNA polymerase* reads a single strand of DNA, one base at a time, while complementary bases are matched. The end result of this transcription is a molecule called *messenger RNA* (mRNA) with the base T replaced by the base Uracil (U).

Transcription Factors RNA polymerase does need guidance about where the transcription should start. The molecule therefore forms a complex with a set of subunits, called general transcription factors. One of these factors recognizes a DNA sequence called the TATA box. The consensus sequence for this TATA box is TATA A/T A A/T. The TATA box is located in the promoter sequence, the noncoding DNA

sequence upstream of a gene, typically 25 base pairs upstream of the transcription start site.

Figure 4.3 shows the transcription initiation complex, which apart from the RNA polymerase and the general transcription factors, also shows the presence of 4 additional components: the mediator, enzymes for modifying histones and chromatin and activator proteins.

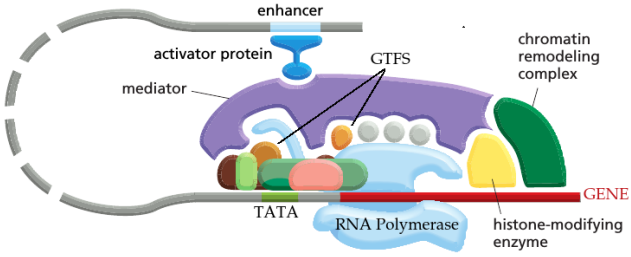


Figure 4.3: Formation of the transcription initiation complex, where RNA polymerase is brought into the position to start reading the gene sequence. RNA polymerase does not work in isolation, it makes use of general transcription factors (GSTFs) but also enzymes for motifying histones and chromatin. The latter enable better access to the gene's DNA. A collection of activator proteins, which bind to the mediator, play an important role in controlling the rate of transcription. Due to the 3D structure of the DNA molecule, in fact enhancer/silencer sites can be thousands of basepairs away from the transcription start site.

Adding Switches for transcription control So far we have not introduced any specificity in the RNA transcribing mechanism. The switches responsible for controlling the rate of gene expression are cis-regulatory sequences. The switches are 'pressed' when a certain protein, called a transcription regulator, binds to these sequences and to the mediator (Figure 4.3).

There are two types of transcription regulators: activators and repressors. *Activators* bind to *enhancer sites* and have a positive impact on the rate of gene transcription. This is accomplished in two ways:

- Activators can help in the recruitment of RNA polymerase and the assembly of the initiation complex.
- Activators can also help in the recruitment of chromatin-modifying enzymes.

Chromatin is made up of *nucleosomes* which consists of a set of *histones* with DNA wrapped tightly around them. In this form chromatin is inaccessible to most regulatory proteins, which explains the need for these enzymes. Typically these enzymes help in expanding the chromatin, making it accessible for TF binding. Note that only part of the DNA is on the outside of chromatin, therefore not all DNA can be bound by TFs.

Repressors bind to *silencer sites* and obstruct the binding of RNA polymerase. This is achieved using multiple mechanisms:

1. Competitive overlap: The silencer partially overlaps with an enhancer site. The bound repressor therefore prevents an activator from binding.
2. Obstructing the creation of the initiation complex.
3. Recruitment of chromatin-remodeling enzymes, which counter-act the actions of chromatin-modifying enzymes.

We should also mention that other mechanisms can play a role in the accessibility of chromatin. For example methylation, the addition of a methyl group to the CpG islands, can also influence the ability of certain transcription factors to bind. Such modifications can again work both activating as repressing.

We now have a clearer view on the mechanisms for turning on a single gene. It is however important to understand that these genes do not act in isolation. In fact, the transcription of one gene could result in a regulatory protein, which might control another gene's transcription. Up to 10% of the genes code for transcription factors, which gives a clear idea on how regulatory control evolves in a complex graph of interactions. Even more complexity is added, if we take into account that each gene has a (unique) set of regulatory sites. Genes, which have a common binding site for a certain regulatory protein, might be involved in the same pathway.

In chapter 6 we will introduce an algorithm for computational motif discovery using comparative genomics. This algorithm is specifically designed to predict new regulatory elements.

For this section we mainly relied on the reference work titled 'Molecular Biology of The Cell'[1] and the PhD of Marleen Claeys [2]. Figures have also been borrowed from this book.

References

[1] B. Alberts. Molecular biology of the cell. Garland science, 2017.

[2] M. Claeys. Probabilistic algorithm for finding motifs in sets of orthologous sequences. PhD thesis. KU Leuven, 2014.

[3] L. Comai. The advantages and disadvantages of being polyploid. *Nature reviews genetics*, 6(11): 836–846, 2005.

[4] J. A. Fawcett, S. Maere, and Y. Van de Peer. Plants with double genomes might have had a better chance to survive the cretaceous–tertiary extinction event. *Proceedings of the National Academy of Sciences*, 106(14):5737–5742, 2009.

[5] M. Garcia-Diaz and T. A. Kunkel. Mechanism of a genetic glissando*: structural biology of indel mutations. *Trends in Biochemical Sciences*, 31(4):206–214, 2006.

[6] M. Hurles. Gene duplication: the genomic trade in spare parts. *PLoS biology*, 2(7):e206, 2004.

[7] N. C. Jones and P. Pevzner. An introduction to bioinformatics algorithms. MIT press, 2004.

[8] S. Ohno. Evolution by gene duplication. Springer-Verlag, 1970.

[9] A. Stark, M. F. Lin, P. Kheradpour, J. S. Pedersen, L. Parts, J. W. Carlson, M. A. Crosby, M. D. Rasmussen, S. Roy, A. N. Deoras, et al. Discovery of functional elements in 12 drosophila genomes using evolutionary signatures. *Nature*, 450(7167):219–232, 2007.

[10] H. Tang, J. E. Bowers, X. Wang, R. Ming, M. Alam, and A. H. Paterson. Synteny and collinearity in plant genomes. *Science*, 320(5875):486–488, 2008.

[11] Y. Van de Peer, J. A. Fawcett, S. Proost, L. Sterck, and K. Vandepoele. The flowering world: a tale of duplications. *Trends in plant science*, 14(12):680–688, 2009.

[12] Y. Van de Peer, S. Maere, and A. Meyer. The evolutionary significance of ancient genome duplications. *Nature Reviews Genetics*, 10(10):725–732, 2009.

[13] J. D. Watson, F. H. Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

Chapter 5

Mining for Homologous Regions

> *One can state, without exaggeration, the observation of and the search for similarities and differences are the basis of all human knowledge.*
>
> —Alfred Nobel.

When genomic regions are *similar* in gene content, this is a strong indication of *homology*, i.e. the existence of a common ancestor. Studying homology between related species and even within a single species, sheds a light on the processes that govern genome evolution.

In this chapter we will introduce an algorithm called i-ADHoRe, which stands for 'iterative and Automatic Detection of Homologous Regions'. More specifically we will focus on version 3 of the algorithm, which shows major improvements in terms of performance and sensitivity with respect to its previous releases. The progress in terms of performance can be attributed to optimizations in the underlying algorithms and data structures, as well as support for running in a parallel computing environment. Furthermore, statistical methods have been updated and a new alignment algorithm has improved the profile-based search for homology.

We choose not to fully enclose the i-ADHoRe research paper [15], but to offer a perspective most interesting to the computer and data scientist. This translates into paying more attention to the algorithmic point-of-view instead of the biological validation.

5.1 Introduction

The field of bioinformatics which focuses on learning from similarities and differences, is called *Comparative Genomics* [3]. In this chapter we are interested in using Comparative Genomics, to gain a deeper understanding into the mechanisms, that allow genomes to evolve. These mechanisms were previously described in section 4.2.

i-ADHoRe is an algorithm for *sequential pattern mining*. The patterns in this context correspond to homologous regions between different gene sequences.

Need for a Big Data approach The detection of homologous regions requires very sensitive and accurate algorithms especially in the case of duplicated regions that have undergone massive gene loss.

The mining of homologous regions might gain in power when more species are compared simultaneously. This does however require software that can handle the associated amounts of 'Big Data' and that can offer (linearly) scalable performance. The data mining algorithms have to support scaling out to a multi-node setting, which typically corresponds to deployment in a data center such as the HPC[1].

How does i-ADHoRe compare to other methods?

1. In contrast to tools that infer genomic homology using WGAs [4, 5, 6, 7], i-ADHoRe detects genomic homology focusing on **patterns in gene lists** and thus ignoring the underlying DNA base pairs.

2. Most algorithms try to detect homology by making pairwise comparisons of the genome sequences. Multi-species collinearity can then be inferred by relying on transitive homology [19]. i-ADHoRe on the other hand makes use of a **profile-based approach** which analyzes **multiple (≥ 2) genomes at once**. Only 2 methods try to find higher level homologous regions, i.e. go beyond pairwise homology. MCScan [24] makes use of transitive homology and Cyntenator [16] makes use of a provided phylogenetic tree imposing an order upon which pairwise comparisons can be made.

3. By focusing on algorithmic **performance** and introducing **parallelization**, i-ADHoRe can analyze a large number of genomes in a very limited timespan.

[1] The Stevin Supercomputer Infrastructure used in part I of this book is provided by the Flemish Supercomputer Center. This infrastructure is funded by Ghent University, FWO and the Flemish Government – department EWI

i-ADHoRe is an *iterative* gene clustering algorithm. In the first iterations high quality constraints are imposed on the initially discovered clusters. These constraints are gradually relaxed, thus ensuring that the strongest patterns are picked up early on as seeds, but at the same time ensuring its ability to detect weaker homology as well. After a first phase where *level-2 Multiplicons* are formed, i-ADHoRe enters a second phase where - again iteratively - the Multiplicons undergo a multiple sequence alignment, to form *profiles*. These profiles are then used, to scan for additional matches in the gene lists. This approach has a higher sensitivity, as patterns are picked up, which might only weakly match with the individual components of the Multiplicon, but match strongly with the *union pattern*. For this reason, i-ADHoRe can also discover patterns, which would not be revealed based on transitive homology.

The concepts 'level-X Multiplicon', 'profile' and 'BaseCluster' are clarified in Figure 5.1. This is identical to Figure 5.10, but there the caption focuses on the flow of the i-ADHoRe algorithm.

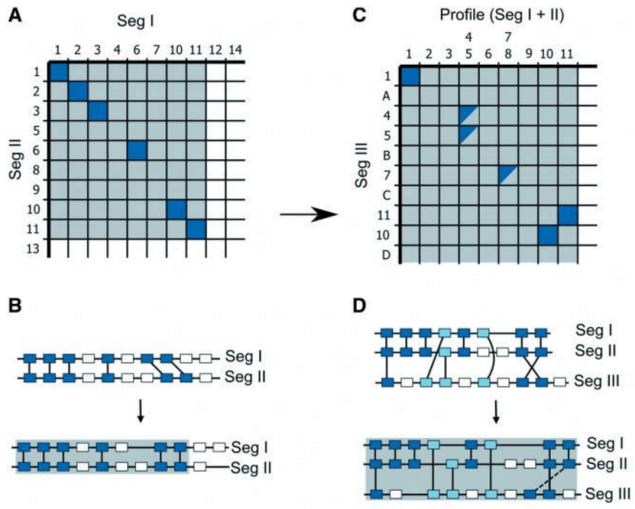


Figure 5.1: (A) A collinear region is detected between two genomic segments. This is called a **BaseCluster**. (B) If the cluster passes the statistical test it is called a **Level-2 Multiplicon**. Then the Multiplicon is aligned to form a **Profile**. (C) The profile is used to scan all genomic segments for additional collinearity. (D) The newly picked up segment forms a **Level-3 Multiplicon**, which is again aligned into a new **Profile**

Compared to the previous version of the algorithm, the level-2 Multiplicons have a higher quality. This is due to the use of an updated statistical approach, taking into account *multiple hypothesis correction*. Also the profiles are much less error

prone than in previous version. This can be attributed to a new multiple sequence alignment algorithm [9], which does not use a progressive approach.

Previous versions of the algorithm were only able to detect collinearity. The new version of the algorithm also detects synteny, albeit limited to *pairwise synteny*.

By optimizing the choices for data structures and search algorithms the new version of i-ADHoRe 3.0 already achieves a gain in speed of a factor 30. With the introduction of the MPI parallelization a speedup factor of 1000 was measured. This gain in performance puts i-ADHoRe in the lead and allows it to analyze some of the most challenging datasets currently available, such as the Ensembl [10] dataset, which upon publication of this work contained 49 eukaryotic genomes. Using a multi-node setup with 8×8 CPU cores, the i-ADHoRe runtime was already reduced to less than 1 hour.

i-ADHoRe is more sensitive than its competition and predecessors. This can be seen in its ability to detect ultra-conserved regions which are detected in over 20 different organisms in the Ensembl dataset. For example, the HOX[2] cluster, which plays a large role in the body plan of organisms, and for which a loss of gene order leads to species with wrongly positioned body parts [12], is detected.

Permutation tests show that the increased sensitivity does not result in a higher False Positive Rate (FPR). It is shown that due to the introduction of multiple hypothesis corrections the FPR is in fact controlled.

The different algorithmic improvements ensure that i-ADHoRe 3.0 will remain a powerful tool to study genome evolution.

Personal Contributions and Project Context

i-ADHoRe 3 is the result of a collaboration between VIB and IDLab. For VIB the most active contributors were Sebastian Proost and Prof. Vandepoele, for IDLab Prof. Fostier and myself took the most active role. Typically the role of the VIB researchers consisted in the biological validation of the algorithm while the IDLab researchers focused on improving the (parallel) performance of the algorithm. My contributions include:

[2] From the corresponding Wikipedia article: Hox genes control the body plan along the head-tail axis. In the example of fruit flies these genes determine the positions of the wings, antennas, limbs. Changing the order of the genes leads to homeotic transformations, with for example a leg occurring at the locus for an antenna.

- Extension of the i-ADHoRe algorithm with a module to detect syntenic regions
- Prof. Fostier pioneered the MPI parallelization of the i-ADHoRe algorithm for collinear regions. I embedded the module to detect synteny in this parallelization scheme.
- Revising, optimizing the 2.0 algorithm in terms of data structures and algorithms in order to speedup algorithms for searching in the GHM. This also included typical developer tasks such as performing unit tests which for example revealed subtle errors in the statistical routines.
- Design of two visualization modules to assess the quality of the gene sequence alignments and the collinear regions in the Gene Homology Matrix (GHM).
- In a separate research paper Prof. Fostier designed a new algorithm for multiple gene sequence alignment in the profile search [9]. Apart from the visualization module I have not contributed to this part.
- The actual i-ADHoRe paper was written by Sebastian Proost, my contribution to the writing process was minimal. This also motivates the fact that this chapter does not contain any literal passages from the paper but instead shows my personal viewpoint on this work.

5.2 Data Structures and Algorithms

The i-ADHoRe algorithm tries to detect patterns in sequential data. Common techniques for addressing problems in *sequential pattern mining*, are string processing algorithms and algorithms for itemset mining.

Multiple Sequence Alignments The standard approach to compare biological sequences is to resort to *Sequence Alignments*. Two textbook algorithms for pairwise alignments are the Needleman Wunsch (NW) algorithm [13] for *global* sequence alignment and the Smith-Waterman algorithm [22] *local* sequence alignments (see info box below).

The NW algorithm has a computational complexity $\mathcal{O}(N^2)$ with N the sequence length. The NW algorithm can be trivially extended for multiple sequence alignments by using an M-dimensional Manhattan grid, but then the complexity becomes $\mathcal{O}(N^M)$. The complexity can be lowered by making use of a greedy progressive

approach starting from the best pairwise alignment and iteratively adding a new sequence, this reduces the computational complexity to $\mathcal{O}(M^2 \cdot N^2)$ and most multiple sequence aligners follow a similar strategy [11].

A severe limitation of progressive alignment approaches is that they are susceptible to bad seeds, i.e. bad initial pairwise alignments. This problem becomes even more pronounced when the *divergence* between the sequences increases.

BACKGROUND: Pairwise Sequence Alignment

The *Needleman-Wunsch* algorithm is an algorithm for global sequence alignment. The optimal pairwise sequence alignment is found by making use of a dynamic programming algorithm for finding the optimal path in a Manhattan grid. To calculate the score at every node $n_{i,j}$ a penalty function $\delta(v_i, w_j)$ is applied. This penalty function has a score for matches $(v_i = w_j)$, mismatches $(v_i \neq w_j)$ and insertions/deletions ($v_i = -$ or $w_j = -$). The NW formula for the score $s_{i,j}$ at $n_{i,j}$ is then:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

To keep track of the optimal path, *back-pointers* are stored at every node. For a visual representation of Manhattan grid and the back-pointers have a look at Figure 5.2.

The *Smith-Waterman* algorithm is a subtle variation on the NW idea. The goal in this case is to retrieve the optimal local sequence alignment.

- At every locus a *source link* is added, which corresponds to adding a '0' in the NW formula.
- The endpoint of the alignment can be any vertex in the Manhattan grid with the highest score.

i-ADHoRE... stands for iterative and Automatic Detection of Homologous Regions. i-ADHoRe takes a different approach to finding homologous regions. The alignment procedure is split into multiple steps:

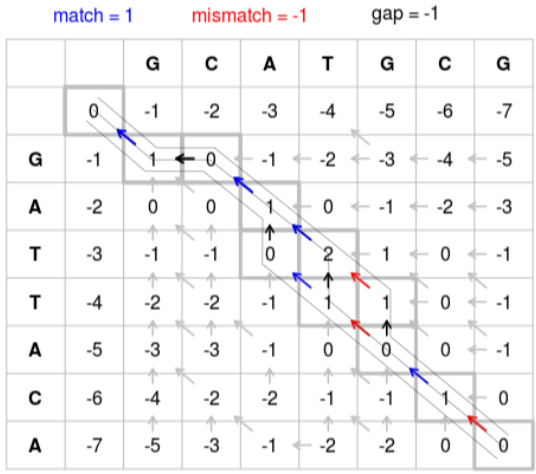


Figure 5.2: Manhattan grid for the global alignment of 2 DNA sequences GATTACA and GCATGCG (from wikimedia.org [21]). Back-pointers corresponding to the optimal path shown.

1. A preprocessing step based on protein alignments for inferring gene homology;
2. A **clustering** algorithm for detecting collinear and/or syntenic regions;
3. A Multiple Sequence Alignment (MSA) of the gene lists corresponding to collinear regions;
4. A **profile search** to scan beyond pairwise homology.

By working with *gene lists* as opposed to DNA base pairs the alignment algorithms are much more robust against noise introduced by sequence divergence, as point mutations will not be visible to the algorithm in the first approach.

The problem can still be thought of as a string problem, albeit with a much larger alphabet. This alphabet then consists of the set of unique pairwise nonhomologous genes.

A clustering algorithm to detect homologous gene lists is more flexible than an MSA. As an example it is possible to detect regions which include inverted segments. The clustering algorithm follows a *greedy* strategy but is *iterative* in the sense that it relaxes the cluster quality constraints after every iteration, thus ensuring that the strongest regions are detected first.

In a parallel research track a *novel alignment algorithm* [9] was designed specifically for gene sequences in order to create profiles. The new algorithm is designed to circumvent the fallacies of progressive alignment approaches. Previous versions of i-ADHoRe [18] relied on a progressive NW aligner.

After aligning the gene lists, *profiles* are built which are used to scan all input gene lists for further (weak) homology. Profile searches have proven their merit in detecting more degenerate genetic homology [18, 20, 29].

The i-ADHoRe algorithm can be run in two different modes which we will discuss in the following sections. First we will describe how i-ADHoRe can detect pairwise collinearity. Next, we will describe how the algorithm was modified to also detect pairwise synteny. After that we will dive deeper into the second phase of the algorithm which is called the profile search. We conclude with a discussion of the parallelization schemes for both stages.

Detecting Pairwise collinearity

The original ADHoRe [28] algorithm detects pairwise homology by scanning for collinear regions in a Gene Homology Matrix (GHM). A GHM is built from a pair of gene lists. A GHM is a sparse dot matrix, as can be seen in Figure 5.3, with the dots corresponding to homologous gene pairs. A collinear region will appear as a diagonal. The dots correspond to a $+1$ in the matrix if the genes have the same orientation or a -1 if their orientation is inverted.

BACKGROUND: BLAST

BLAST stands for **B**asic **L**ocal **A**lignment **S**earch **T**ool. It is used for searching for similar sequences (given a query sequence) within a sequence database. Without going into further details we can state that the algorithm consists of a method based on K-gram similarity followed by a local alignment step. Alignments that pass a threshold score are returned. BLAST is one of the most highly cited papers in bioinformatics with over 50,000 citations.

Preprocessing steps The input data for the i-ADHoRe algorithm is depicted in Figure 5.4. The datasets are not challenging in terms of size, with a genome typically only requiring a storage size in the order of megabytes.

The data consists of a number of gene lists, typically one per chromosome and a blast table with a set of gene pairs which are the result of an all-against-all similarity

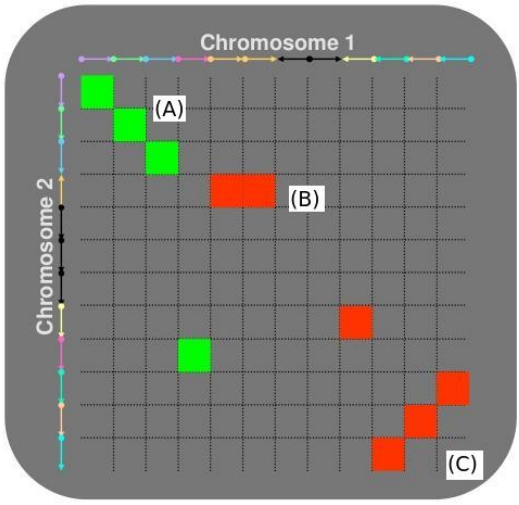


Figure 5.3: Gene Homology Matrix for two chromosomes with equally oriented genes (green) and inversely oriented genes (red). Region (A) and (C) are candidate collinear regions while (B) shows a tandem duplication. (from Vandepoele [26])

Figure 5.4: The input of the i-ADHoRe algorithm is a configuration file which lists a collection of genomes, each corresponding to a number of gene list files. A gene list file contains the genes of genome segment, as an ordered sequence, together with the orientation of the genes. The blast table consists of two columns. These columns correspond to homologous gene pairs or to genes together with a gene family identifier.

search of protein sequences using BLASTP [1]. The corresponding genes of two protein sequences with more than 30% sequence identity[3] are considered homologs. For short protein sequences the HSSP score is used [17].

As can be seen in Figure 5.3, tandem duplicated genes potentially distort the diagonal regions in the GHM. Tandem duplications are therefore removed by mapping them to the closest homolog with the same orientation, such that the first of the genes serves as the representative. (the maximal 'tandem gap' distance for remapping can be configured).

The iterative clustering scheme is split in two separate parts, one for each orientation class (see also Fig 5.3).

For a satellite view on the ADHoRe algorithm, have a look at Figure 5.5.

Greedy clustering For each orientation class the iterative clustering algorithm is run. In each iteration the GHM is scanned (multiple times) in a row-major order. Enforcing the order turns this into a *greedy* algorithm: clusters are picked up if they meet certain quality constraints. There are situations where this can lead to a suboptimal clustering.

> **Example**: Imagine a collinear region consisting of 2 segments, a short segment S_1 and a long (optimal) segment S_2. Both segments have a slightly different slope. Since S_1 is detected first it might also take in dots of S_2 as long as the cluster quality constraints (see later) are met. This might however make it more difficult to pick up the remainder of S_2.

Each iteration i of this greedy clustering algorithm is controlled by an increasing *gap size* g_i, going from $g_0 = 3$ to maximum gap size G in the tenth iteration: $g_9 = G$. During each iteration g_i is an upper boundary for the distance between two points in a cluster. The gap sizes are equally spaced on a log scale[4]. One iteration consists of 3 steps:

1. **Formation of BaseClusters:** We iterate over all *singletons* (= dots not part of a cluster). For each singleton the vicinity is scanned for additional dots within g_i. For each available singleton this process is repeated until no more matches can be found. Next, a BaseCluster is created if (i) at least 3 dots (the number can be configured) are found and (ii) the quality of the diagonal region satisfies a quality constraint Q_1. The BaseCluster is then stored and its dots

[3] Sequence identity is defined as one minus the fraction of gaps in a multiple sequence alignment.
[4] An example of a log scale going from 1 to 100 might correspond to: $[1, 3, 10, 32, 100]$

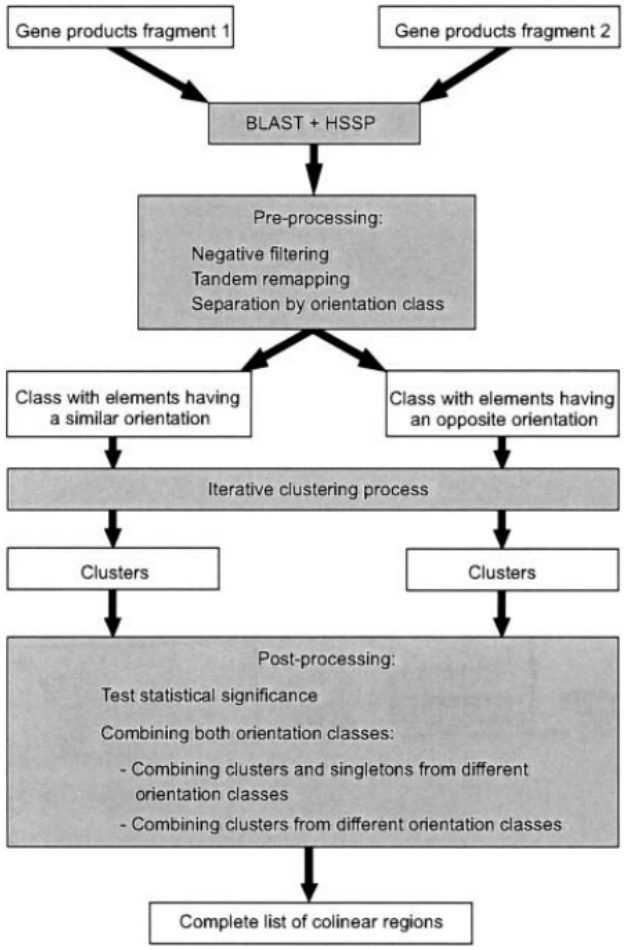


Figure 5.5: High-level overview of the ADHoRe algorithm for detecting pairwise homology. (from Vandepoele [28])

(called *Anchor Points* (AP)) are masked (=removed from the list of singletons). The process repeats until all singletons have been tested.

2. **Enrichment of BaseClusters with Singletons:** Instead of trying to form new BaseClusters, singletons can also be added to one of the already detected clusters. A singleton is added, if all of the following conditions are met: (i) the dot lies within g_i of the cluster begin and endpoint or is located within its bounding box (ii) the new dot lies in the confidence interval of the BaseCluster (Q_2) and (iii) the new candidate cluster still satisfies Q_1.

3. **Merging of nearby BaseClusters:** For every BaseClusters the distance to the other clusters is calculated. (The distance is calculated between the cluster centers) Clusters that lie within g_i' (there's a separate parameter for cluster distances $g' \geq g$) are candidates for merging. Similar criteria as in step 2 apply: If the clusters overlap Q_2 should apply for the dots in the overlapping range, to avoid closely parallel clusters being joined. Prior to the merge it is verified that also Q_1 holds for the joined cluster.

Each time an AP is added or two clusters are merged the statistics concerning the regression line are updated.

Diagonal Pseudo Distance From the viewpoint of every singleton, quadrant IV of the coordinate plane is scanned for new dots. If a dot is found within the current g_i, it is added to the cluster. The ideal cluster however has its dots along a line with slope $a = -1$. The algorithm is greedy and always adds the closest point, points on slope $a = -1$ should therefore be prioritized. This is achieved by introducing a custom distance measure, the *diagonal pseudo-distance* (dpd):

$$d = 2\max(|y_2 - y_1|, |x_2 - x_1|) - \min(|y_2 - y_1|, |x_2 - x_1|)$$

Note that we use 'pseudo-distance' to indicate that this transformation does not satisfy the triangle inequality:

> **Triangle inequality violation:** Choose 3 points and their corresponding coordinates: A(0,0), B(2,0), C(1,1). Then the triangle inequality should hold:
>
> $$d(A,B) <= d(A,C) + d(B,C)$$
>
> But $d(A,B) = 4$ and $d(A,C) + d(B,C) = 2$

A visual representation of the dpd can be seen in Figure 5.6.

Cluster Quality Measures So far 2 quality measures have been used to accept or reject a BaseCluster: Q_1 and Q_2. Q_1 corresponds to the *goodness-of-fit* of the regression line through the dots:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_j (y_j - \bar{y})^2} \tag{5.1}$$

Here y_i correspond to the actual data points, $\hat{y}_i$ are the values predicted by the regression line, and $\bar{y}$ correspond to the average value for y. The coefficient of determination R^2 corresponds to the fraction of the variance in the data that can

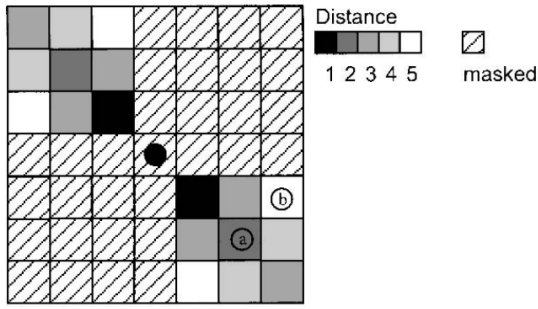


Figure 5.6: Diagonal Pseudo-distance for two gene lists with the same orientation class. Color scale indicates the distance to all points. Distance $d_a = 2 \cdot 2 - 2 = 2$, $d_b = 2 \cdot 3 - 1 = 5$. The pseudo-distance increases slower if the absolute value of the slope is closer to 1.

be explained by the model. The minimum value for R^2 can be configured (default $q_{val} = 0.9$).

Q_2 plays a role when a singleton is added or overlapping clusters are merged. In order for a singleton to be added it should fall in the *prediction interval* of the regression model (Note: the prediction interval is wider than the *confidence interval*). For a new value x^* the interval for y is given by:

$$\hat{y} \pm t^*_{n-2} s_y \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{(n-1) \cdot s_x^2}} \tag{5.2}$$

with $\hat{y}$ the y-value on the regression line, t^* corresponding with the student t-distribution ($\alpha = 0.01$), s the sample variance, $\bar{x}$ the sample average and n the number the dots in the cluster.

Cluster Significance Once the iterative algorithm comes to a stop, we are left with a set of putative BaseClusters. Every cluster in the set is now evaluated in terms of its probability of being generated by chance (given the number of dots in the GHM).

The *local* p-value of a cluster is the probability of finding a certain cluster within its bounding box. p_{loc} of a cluster is calculated as the product of probabilities over all dots of finding every next dot by chance. For a dot a_n preceded by a BaseCluster $a_{n-1} \ldots a_1$ this probability is given by:

$$p_{loc}(a_n | a_{n-1} \ldots a_1) = p_{loc}(a_n) \cdot p_{loc}(a_{n-1} \ldots a_1) = \rho \cdot S_{box} \cdot p_{loc}(a_{n-1} \ldots a_1)$$

with ρ the density of dots in the GHM and S_{box} is the size of the rectangle with corners a_{n-1} and a_n.

Note that this local p-value is always decreasing since every $p_{loc} \leq 1$, furthermore p_{loc} is independent of the dimensions of the GHM. This conflicts with the observation that in a GHM with infinite size every possible cluster, however unlikely, should be observed.

The actual global p-value p_{glob} of a cluster is the probability that, given a GHM, a certain cluster C occurs at least $X \geq 1$ times by chance. The maximal number of clusters that can be formed depends on the number of AP N in the GHM and is therefore the probability that none of them results in this cluster C:

$$p_{glob} = P(X \geq 1) = 1 - P(X = 0) = 1 - (1 - p_{loc})^N$$

The global p-value per cluster is then compared with a cutoff value 'prob_cutoff' and the clusters not satisfying this threshold are rejected and there dots unmasked.

Fitting a regression line to a set of dots corresponds to making a hypothesis. Since many hypotheses are generated from the same GHM a *Multiple hypothesis correction* has to be performed (see infobox below). The i-ADHoRe algorithm supports both the Bonferroni correction [2] as the False Discovery Rate [8]. The first correction is the more conservative approach. More information on multiple hypothesis testing in the box on page 84.

Merging orientation classes In a final step clusters and singletons from different orientation classes can be combined. In this step a *Level-2-Multiplicon* is formed, which is a collection of one or more BaseClusters possible from different orientation classes. The distance between these clusters is calculated between the cluster centers. Cluster quality metrics are applied after twisting the clusters from the -1 orientation. This final step allows for the detection of larger collinear regions which also consist of small gene inversions. Level-2-Multiplicons are aligned and thus form a *Profile*. Originally, the corresponding gene lists in a profile were aligned by applying the Needleman-Wunsch (NW) Algorithm in a progressive matter [11]. The number of gaps in the alignment is kept track of as a quality measure, with too many gaps a Multiplicon is no longer used for further profile searches.

Complexity/Performance Details To analyze the scalability of the i-ADHoRe algorithm we need to analyze the different steps of the algorithm separately. The i-ADHoRe algorithm consists of 3 different algorithmic steps: the *clustering* step, the *alignment* step and the *profile search*.

We will first analyze the algorithm for detecting collinear regions. The greedy clustering algorithm for detecting BaseClusters has complexity $\mathcal{O}(N^3)$, with N the length of the longest gene list, although in practice the $\mathcal{O}(N^2)$ segment of the algorithm dominates. Aligning multiple gene lists also has complexity $\mathcal{O}(N^2)$, similar to the progressive alignment approaches as discussed on page 74. The profile searches are a combination of the greedy clustering algorithm and a Branch-and-Bound algorithm to build more complex Multiplicons up until a maximum number of gaps in the alignment has been reached.

BACKGROUND: Multiple Hypothesis Testing

When evaluating the same hypothesis multiple times, the probability of observing a rare event increases.

> **Example:** if we try throwing dice, for a long enough sequence, we will finally have an experiment where N consecutive throws result in a 6.

The probability of having at least one false positive is called the *Family Wise Error Rate* ($FWER$). In order to control

$$FWER \leq \alpha$$

The significance levels of the m individual hypotheses H_i have to be more conservative. To satisfy the $FWER$ condition we can apply the *Bonferroni Correction* requiring the p-value p_i of H_i to satisfy:

$$p_i \leq \frac{\alpha}{m}$$

A less stringent restriction is the *False Discovery Rate* (FDR) which tries to keep the expected proportion of false positives below α.
In the Benjamini-Hochberg procedure we sort the individual p-values in ascending order and find the largest k for which:

$$\max_k p_k \leq \frac{k}{m}\alpha$$

The following null hypotheses (from $k+1 \mapsto m$) are then rejected.

Complexity of the pair-wise algorithm Let's further analyze the complexity of the pairwise algorithm. In doing so we will always consider two extreme scenarios: a weak **scenario I** where $\mathcal{O}(N)$ short BaseClusters can be formed and a strong **scenario II** where the two gene lists are completely collinear and therefore form $\mathcal{O}(1)$ long clusters.

A GHM is a sparse matrix, with $\mathcal{O}(N)$ non-null elements and dimensions $N \times N$. The iterative aspect of the algorithm can be ignored since the number of iterations is a constant.

Data Structure: GHM

A GHM is implemented as a sorted map, with the values being a sorted set of integers corresponding to the positions of the APs. Constructing a GHM requires $\mathcal{O}(N \log N)$ operations.

Data Structure: BaseCluster

A BaseCluster is *multiset* of anchor points, sorted by X-coordinate.

The algorithm consists of a number of basic building blocks which are often re-used:

1. *Scan*: Iterate over the GHM while calculating distances dots in a certain range. Due to the sorted nature of the GHM this operation has complexity $\mathcal{O}(N \log N)$.

2. *Add*: Adding a dot to a BaseCluster requires $\mathcal{O}(\log N)$ operations per dot. In scenario II adding all dots to a cluster has complexity $\mathcal{O}(N \log N)$.

3. *Fit*: When a dot is added the regression coefficients have to be updated:

$$\begin{aligned} \beta_1 &= \frac{E(XY) - E(X)E(Y)}{Var(X)} \\ \beta_0 &= E(Y) - \beta_1 E(X) \end{aligned} \tag{5.3}$$

   $E(\cdots)$ corresponds to the expected value of a variable. The regression line is defined as $Y = \beta_0 + \beta_1 X$. In scenario I this corresponds to a total cost of $\mathcal{O}(N^2)$

4. *Check*: The quality checks Q_1 (eq. 5.1) and Q_2 (eq. 5.2) contain a summation over all points in the cluster. For scenario II this implies a $\mathcal{O}(N^2)$ workload.

5. *Merge*: The actual merging of two clusters (including singletons and clusters) corresponds to 2N add operations. The merging algorithm is very similar to

that of *hierarchical clustering*, where at each iteration the 2 closest clusters are joined. Without any optimizations this has a complexity of $\mathcal{O}(N^3)$ for scenario I.

The *resulting complexity* of the algorithm is therefore $\mathcal{O}(N^3)$. In reality the number of clusters is much smaller than the number of dots. Therefore the runtime is in practice dominated by the $\mathcal{O}(N^2)$ term.

Possible optimizations In case the GHM processing becomes the bottleneck of the algorithm it is possible to further optimize the individual building blocks.

- The *Fit* operation can be lowered to $\mathcal{O}(N)$ by incrementally computing the averages and the variance. ($Var(X) = E(X^2) - E(X)^2$)
- The *Check* operation can be sped up relying on an approximation, thereby assuming β_1 and β_0 are constants.
- The *Merge* operation can be trivially transformed to $\mathcal{O}(N^2)$ by only calculating the cluster distances once and storing them in memory.
- The complexity of the *Merge* operation can be further lowered by representing the clusters by a number of reference points. The nearby clusters are then found by relying on the *Scan* operation to look for the nearest clusters, lowering the complexity to $\mathcal{O}(N \log N)$.

Detecting Pairwise synteny

Collinearity can be more easily detected in between closely related species or in the case of more 'recent' duplication events. Syntenic regions have a shared gene content, but do not necessarily preserve gene order.

i-ADHoRe can be run in *synteny* mode and in *hybrid* mode. Both modes are restricted to the detection of pairwise homology, as bags of genes cannot be aligned. In the hybrid mode the GHM is scanned for collinear regions and after masking their APs the synteny mode is run.

Synteny versus Collinearity In Figure 5.7 the differences between a BaseCluster and a SyntenicCloud are shown. The *bounding box* is important for the efficient implementation of a SyntenicCloud as it simplifies distance calculations (see page 87)

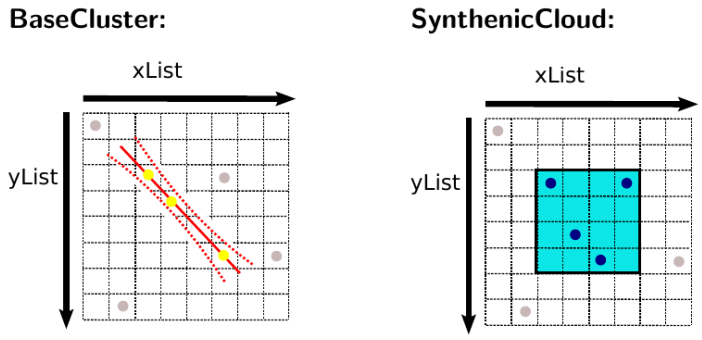


Figure 5.7: Whereas a BaseCluster is defined by a regression line and a prediction interval, a SyntenicCloud is a bag of APs inside a bounding box (blue)

In the next paragraphs we will revisit aspects of the algorithm and highlight the differences between both modes.

Greedy Clustering Just as in the collinear mode the greedy clustering algorithm consists of three steps corresponding to the formation, enrichment and merging of SyntenicClouds. An important difference is that there are no quality measures Q_1 and Q_2 for clouds. Rejection only occurs after assessing the cloud significance. To merge 2 clouds we rely on single-linkage hierarchical clustering.

Chebyshev Distance There is no directional preference for cloud structures. Therefore we use the 2D Chebyshev distance (L_∞ metric):

$$d = \max\left(|x_2 - x_1|, |y_2 - y_1|\right)$$

For this metric all dots on a square lie within the same distance of the center.

Cloud Significance The local p-value of a cloud can be calculated using two approaches which rely on the binomial distribution. For a bounding box with dimensions $a \times b$ the probability of detecting a cluster with c or more APs is given by the cumulative distribution function:

$$p_{loc} = P(X \geq c) = \sum_{m=c}^{ab} \binom{ab}{m} p^m (1-p)^{ab-m}$$

with $p = \rho$, i.e. the AP density in the GHM. This p_{loc} is slightly too conservative in the sense that it doesn't take into account the fact that the tandem duplications have

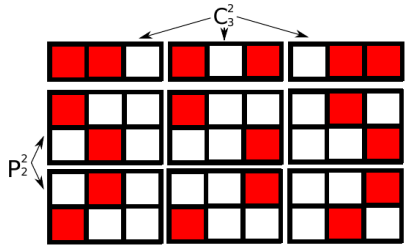


Figure 5.8: Number of possible size $m = 2$ clouds in a 2×3 bounding box taking into account the removal of tandem duplicates is calculated as: $C(2,3) \cdot P(2,2) = 6$.

been removed. A second significance measure takes this into account. The number of possibilities is a product between a partial permutation P and a combination C as is visually depicted in Figure 5.8, leading to the following corrected p-value:

$$p_{loc} = P(X \geq c) = \frac{1}{N_f} \sum_{m=c}^{\min(a,b)} \frac{m!}{(m-a)! \cdot (m-b)!} p^m (1-p)^{ab-m}$$

with a renormalization factor $N_f = P(X \leq \min(a,b))$.

> **Example renormalization**: What is the probability of throwing 4 and 5 with two dies, given the constraint that throwing the same number of eyes is not allowed?
> In this situation $p_{45} = \frac{1}{30}$ which can be obtained via renormalization while omitting $p_{11}, \ldots p_{66}$.

Note that for very large clouds ($\min(a,b) >$ tandem_gap) there might in fact be a small error since for these there might be multiple APs per column/row.

The same multiple hypothesis corrections are used as in the collinear mode.

Merging orientation classes In the synteny mode a GHM is not separated in orientation classes. Instead, the direction of the genes is ignored.

Complexity/Performance Details The unordered nature of a cloud does complicate the distance calculations. Therefore the algorithm be configured to use brute force calculations to obtain an exact distance or to rely on optimizations.

Data Structure: SyntenicCloud

A SyntenicCloud is implemented as a *vector*, with an interface similar to the Java ArrayList. Apart from the 'bag' of APs, a *bounding box* is stored ($2 \times (x,y)$ coordinates).

The distance from a dot to a cloud is defined as the minimum of the distances to all individual APs contained in the cloud. In scenario II this means it takes $\mathcal{O}(N)$ operations to add a dot. Instead we can make use of the cloud's *bounding box* and calculate the distances w.r.t. its corners. This might introduce an error, but it is limited due to the nature of L_∞ metric.

In what follows we focus on the inexact but optimized distance calculations.

We revisit the building blocks of the collinear algorithm below:

1. *Scan*: This operation is the same as for collinear search, with the exception of the region which is scanned for additional APs. This region takes the form of a *frame with thickness* the gap size. The complexity is however unchanged. APs that fall within this frame are added without an actual distance calculation, on top of that multiple APs can be added simultaneously. The latter is due to the absence of quality checks.

2. *Add*: Adding a dot to a cloud requires $\mathcal{O}(1)$ operations, since no order is preserved within the container. This leads to a total complexity of $\mathcal{O}(N)$.

3. *Fit*: When a dot is added, only the bounding box has to be updated which is a $\mathcal{O}(1)$ operation

4. *Check*: No quality checks are performed, except for the final p-value calculation which is an $\mathcal{O}(N)$ operation

5. *Merge*: The actual merging of two clusters, is an $\mathcal{O}(N)$ operation. Again, no quality checks are performed. Therefore, clouds, that lie within the gap size, are merged. This leads to a $\mathcal{O}(N^2)$ cloud comparisons. The distance calculation between 2 clouds is more expensive than in the collinear case. In the brute force case the distance also costs a $\mathcal{O}(N^2)$, bringing the total cost to $\mathcal{O}(N^4)$. This can be immediately lowered to $\mathcal{O}(N^3)$ by only calculating the distance between APs within a distance g_i of the bounding box of both clouds ($\mathcal{O}(N)$ operations), as demonstrated in the right panel of Figure 5.9. We can try to avoid this operation by only performing it when (i) we are certain that the clouds do not overlap or (ii) we are certain that the clouds are not too far away $d \gg g_i$. Case (i) is shown in the left panel of Figure 5.9. There we simply verify whether there are APs in the intersection of the two boxes, although worst case this is still an $\mathcal{O}(N)$ operation. Case (ii) is the most common case and

requires only $\mathcal{O}(1)$ operations. Here a lower boundary for the cloud distance is calculated by calculating all 16 distances between the corners of the bounding boxes. As can be seen in the middle panel of Figure 5.9, the corner distance d_c can be maximally $\frac{s_{max}}{2}$ if the clouds are within g_i. Therefore if $d_c > \max(g_i, \frac{s_{max}}{2})$ the actual distance calculation can be skipped. If we assume that on average only a constant number of clouds is close (this is obvious from a geometrical perspective), the *Merge* operation should scale as $\mathcal{T}(N^2)$

Summarizing the complexity for the above operations we have a complexity of $\mathcal{O}(N^3)$, but amortized this becomes $\mathcal{T}(N^2)$.

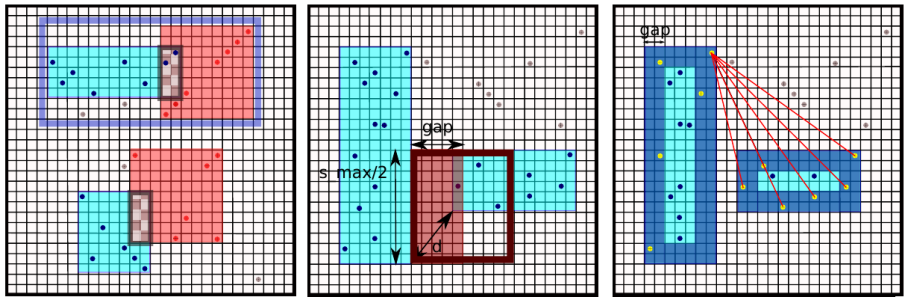


Figure 5.9: Left: Joining two clouds (red and blue) with overlapping bounding boxes. Center: Distance estimation for nearby clouds. Right: Cloud distance calculation only using edge points

Amortized Analysis

Instead of focusing exclusively on the worst case scenario for analyzing the complexity $\mathcal{O}$ of an operation, amortized analysis averages the cost to have a more realistic estimation $\mathcal{T}$ of the true cost of an algorithm. The simplest example is the vector container, which has a $\mathcal{O}(N)$ worst case cost to insert an element, while the amortized cost is $\mathcal{T}(1)$.

Note that the maximum gap parameter must be chosen carefully. Otherwise, this might cause an *avalanche* effect, in which the bounding box keeps growing because new dots are found in the window frame. The latter becomes more problematic, as the cloud expands as also the frame size grows with the radius of a cloud.

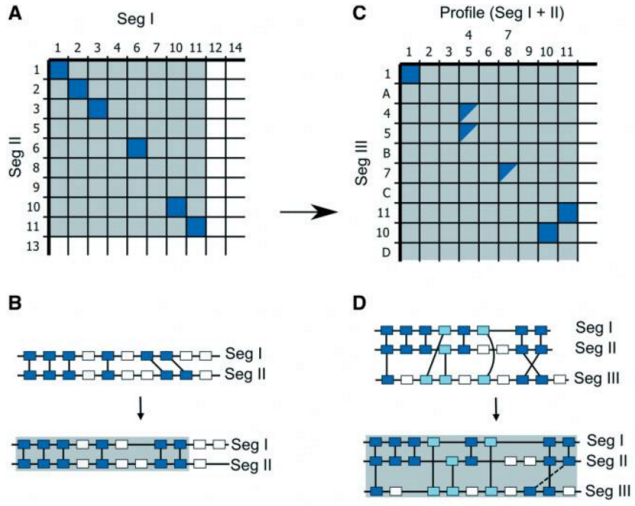


Figure 5.10: (A) A collinear region is detected between two genomic segments. (B) If the cluster passes the statistical test the sequences are aligned, which results in a profile. (C) The profile is used to scan all genomic segments for additional collinearity. The collinear regions detected on the left matches partially with both gene lists but would not be detected by means transitive homology. (D) The 3 gene segments are aligned.

Detecting higher-level collinearity with Profile Search

Figure 5.10 shows the life cycle of a 'successful' Multiplicon. Previously we discussed the detection of pairwise collinearity leading to level-2 Multiplicons. Multiplicons passing the significance tests are aligned and used as *profiles* to detect higher order collinearity, as can be seen in the right-hand side of Figure 5.10.

The second phase of the algorithm consists of matching a profile with the (unmasked) gene segments. If collinearity is detected between a level n Multiplicon and a gene segment, this is again followed by an alignment and the formation of a level $n+1$ Multiplicon.

The alignments are performed with the novel greedy graph-based algorithm designed by Jan Fostier [9].

The search for higher-level Multiplicons is an iterative process: In each iteration the *most promising* Multiplicon is matched against all gene segments. This matching process is very similar to the detection of pairwise homology.

GHMProfiles instead of GHMs The main difference with the detection of pairwise collinearity is, the replacement of the GHM as the data structure to be searched, by a GHMProfile. This data structure inherits from a GHM, but one of the segments can now be a profile, which is the result of aligning multiple gene segments. Since an alignment can contain mismatches, certain positions in the profile can nonhomologous genes. Therefore, partial matches are possible, as demonstrated in Figure 5.10 by partially filled boxes. Because of these partial matches, collinearity in a GHMProfile can be detected which would go unnoticed when using only transitive homology: the third segment can only be detected by search for APs that are homologous to the profile, but a match against the individual segments making up the profile would not lead to a clear collinear cluster.

The remainder algorithm for detecting collinearity is identical to that of pairwise collinearity.

Multiplicon Prioritization The search procedure works using in a priority queue of Multiplicons. The highest priority is assigned to the Multiplicons with the highest level and the highest number of APs. The top Multiplicon is removed from the queue once no more homologous segments can be detected. The segments contained within the Multiplicons are masked.

For a single Multiplicon all matches can be processed in parallel. Parallelization will be discussed in the next section. If multiple segments are found in a single iteration for a level n Multiplicon this results in two separate level $n+1$ Multiplicons.

The algorithm proceeds until the queue contains no more unprocessed Multiplicons.

Parallelization with MPI

The i-ADHoRe algorithm is parallelized, to be able to handle an increasing number of gene list comparisons. As parallelization comes with a certain communication overhead it is important to divide the work in chunks which correspond to a processing time which is larger than this overhead. In this context the first avenue for parallelization is to consider running i-ADHoRe on a single GHM as a unit of work. Parallelization within a single GHM is less trivial due to the inherent sequential nature of the scanning algorithm.

In essence, the parallelization strategy follows a *master-slave scheme*. The role of the master however, is only slightly different from that of a slave: additional tasks

consist of progress logging, collecting all the output of the slave nodes, and write the collected output to disk.

The parallel implementation supports both multiprocessing using MPI and at the same time multithreading via POSIX Threads[5].

In what follows we will focus on the MPI parallelization. Every process can however also run with multiple threads. Since threads share memory, the implementation has to take care of *race conditions*[6]: different threads can simultaneously try to modify the same information. In case of i-ADHoRe, this comes into play if for example 2 different threads try to simultaneously insert a new cluster at position i in a storage vector.

What MPI routines? The only MPI communication functionality needed by i-ADHoRe is the `MPI_Allgatherv` routine[7]. This routine is used to synchronize the set of Multiplicons discovered by the individual processes between the nodes. The static load balancing scheme only requires access to the nodes' rank.

Parallelization level-2-algorithm Typically the input of the i-ADHoRe algorithm consists of N gene lists. To infer homology between the individual gene lists, $\binom{N}{2}$ comparisons have to be made, corresponding to scanning one GHM each. There are no dependencies between these GHMs, therefore they can be processed in separate processes without any communication, making this parallelization *embarrassingly parallel*. The latter implies that a linear speedup can be achieved.

The level-2 parallelization consists of assigning work packages to the different processes and threads. This assignment is static and deterministic: the tasks for the threads are not modified at runtime, each time a thread finishes a task and 'asks for some work' the next tasks will always be the same.

In order to achieve this task partitioning, every GHM is assigned a weight w, according to the lengths of the gene lists l involved. The total workload is then:

$$W_{tot} = \sum_{i=0}^{N} \sum_{j=0}^{i} |l_i||l_j|$$

In order to have a proper *load balancing* we try to assign each process an equal share of the work. $W_{equal} = \frac{W_{tot}}{N_{proc} \cdot N_{thread}}$. We use a parameter W_{min} in the load balancing

[5] https://en.wikipedia.org/wiki/POSIX_Threads
[6] https://stackoverflow.com/questions/34510/what-is-a-race-condition
[7] https://www.mpich.org/static/docs/v3.2/www3/MPI_Allgatherv.html

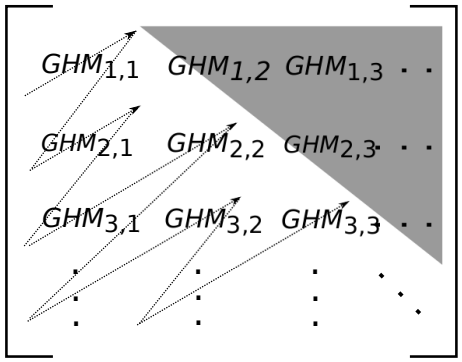


Figure 5.11: Order in which to process the GHMs is indicated with the arrows. Indices correspond to the gene lists in sorted order (long to short). The grey area is not explored due to symmetry.

scheme to determine whether a work package contains sufficient tasks. W_{min} is a fraction of W_{equal}, therefore multiple (small) GHMs might be processed consecutively. The task assignment proceeds by iterating over the possible GHMs as demonstrated in Figure 5.11. This way of iterating is a heuristic approach to approximately rank the workloads in descending order, which avoids building a priority queue. At every step in the iteration the process and thread is selected which has the lowest current (temporary) workload. If this process and thread correspond to the *active thread*, this workload is executed. Otherwise the iteration continues, until the process matches the current process. The iteration ends, when no more work packages are available for the current process. Then all processes synchronize, all local Multiplicons are packed into a buffer and the `MPI_Allgatherv` routine is used to communicate all Multiplicons to the other worker nodes.

Parallelization higher-level algorithm A profile search also consists of N pairwise comparisons with the individual gene lists. Note that only one profile is processed at the same time, which corresponds to $1 \times N$ GHMProfiles. The same load balancing scheme can be applied as for the level-2 algorithm. The main differences are that a profile is in general smaller than a gene list and that there are only N comparisons instead of $\binom{N}{2}$. These differences have an impact on the effectiveness of the parallelization: a good load balancing is more difficult to achieve.

The static load-balancing scheme described here is not necessarily the optimal approach. However, the analysis of the parallel speedup on page 98 and in Figure 5.15

demonstrates that this approach already leads to a high parallel efficiency, with some room for improvement for the higher-level algorithm.

5.3 Results

We will conclude with a short result section in which we want to:

1. Demonstrate the validity of the method from a statistical point of view by comparing p-values with the actual empirical false positive rates.
2. Demonstrate that the algorithm is more sensitive than its competitors and show that it retrieves important biological results, such as the HOX cluster.
3. Prove that the parallelization is scalable and that i-ADHoRe can handle larger datasets more easily than its competitors.

More detailed (biological) results can be found in Proost [14].

False Positive Rate is under control

The p-value of a null hypothesis is an estimate for the False Positive (FP) rate, where the null hypothesis is incorrectly rejected. In the i-ADHoRe algorithm that corresponds to the fraction of detected clusters, that passed the significance tests, but which can still be attributed to chance.

BACKGROUND: Permutation Test

A nonparametric statistical test by which the distribution of a test statistic is estimated by making use *resampling*. More specifically the labels of the data points are shuffled a number of times. For each shuffle the test statistic is calculated, these values then approximate its distribution.

Collinear mode To estimate the FP-rate we made use of permutation testing (see info box above). In this case 100 random datasets were generated from the original *Arabidopsis thaliana* dataset. The gene lists remain constant in size but the genes are randomly re-assigned. The FP-rate was calculated for different combinations of the gap parameters G and the goodness-of-fit quality measure q_{val}. The results for the full parameter landscape are shown in Table 5.1. To achieve p-values of 10^{-2} and less, the most flexible parameter setting would therefore be to use $G \leq 30$ and $q_{val} \geq 0.5$.

Table 5.1: Measuring the Empirical False Positive Rate for collinear clusters using 100 permutation tests generated from the *Arabidopsis thaliana* dataset. Bold-faced values correspond to FP-rates below the 10^{-2} threshold.

G	$q_{val} = 0.5$	$q_{val} = 0.6$	$q_{val} = 0.7$	$q_{val} = 0.8$	$q_{val} = 0.9$	$q_{val} = 1.0$
15	**2,74E-04**	**2,74E-04**	**2,74E-04**	**2,75E-04**	**3,33E-04**	**0,00E+00**
20	**1,33E-03**	**1,34E-03**	**1,40E-03**	**1,39E-03**	**1,52E-03**	**0,00E+00**
25	**4,72E-03**	**4,70E-03**	**4,54E-03**	**4,58E-03**	**4,33E-03**	**0,00E+00**
30	1,02E-02	**1,00E-02**	**9,84E-03**	**9,21E-03**	**9,36E-03**	**0,00E+00**
35	2,17E-02	2,10E-02	1,98E-02	1,86E-02	1,81E-02	**0,00E+00**
40	3,97E-02	3,76E-02	3,50E-02	3,22E-02	2,89E-02	**0,00E+00**
45	7,43E-02	7,00E-02	6,53E-02	5,92E-02	5,01E-02	**0,00E+00**
50	1,27E-01	1,18E-01	1,11E-01	9,94E-02	8,36E-02	**0,00E+00**
55	1,97E-01	1,81E-01	1,65E-01	1,46E-01	1,21E-01	**0,00E+00**

Synteny mode The synteny mode has less parameters to control the quality of clusters, therefore a more stringent parameter landscape is to be expected. The empirical FP-rates are estimated in the same manner, but this time the *human* dataset is used. The human dataset was chosen due to the presence of paralogous regions, originating from an ancient WGD in vertebrates [27] (350-450 million years ago).

The results are shown in Table 5.2. To achieve p-values of 10^{-2} and less, the cloud gap size must be chosen $G \leq 20$. Note that in both sets of tests the gap parameters for merging two clusters were chosen to be $g' = G + 5$. With these parameter settings the i-ADHoRe algorithm was run on the human dataset. Given the ancient WGD, it was expected that the synteny mode would detect additional homologous regions. The number of APs, i.e. points in a cluster, is 544 in the collinear mode, while the synteny mode retrieves 2215 APs. This result encourages the use of the synteny mode, to detect highly diverged homologous regions.

The statistical testing procedures were completely reworked in i-ADHoRe 3.0 compared to prior versions of the algorithm. The effect of introducing multiple hypothesis corrections is shown in Figure 5.12. With this correction in place the empirical FP-rate is improved upon by a factor 10 as compared to the previous version of the algorithm and are much closer to the idea curve (red). To achieve an empirical FP-rate $\leq 5\%$ p-values should be chosen in the range $p \in [10^{-3}, 10^{-1}]$.

Table 5.2: Measuring the Empirical False Positive Rate for the synteny mode using 100 permutation tests generated from the *human* dataset. Bold-faced values correspond to FP-rates below the 10^{-2} threshold.

G	FP-rate
5	**4.27E-04**
10	**5.25E-03**
15	**2,14E-02**
20	5,42E-02
25	1,06E-01
30	1,78E-01
35	2,77E-01
40	3,98E-01
45	5,22E-01
50	6,47E-01
55	7,90E-01

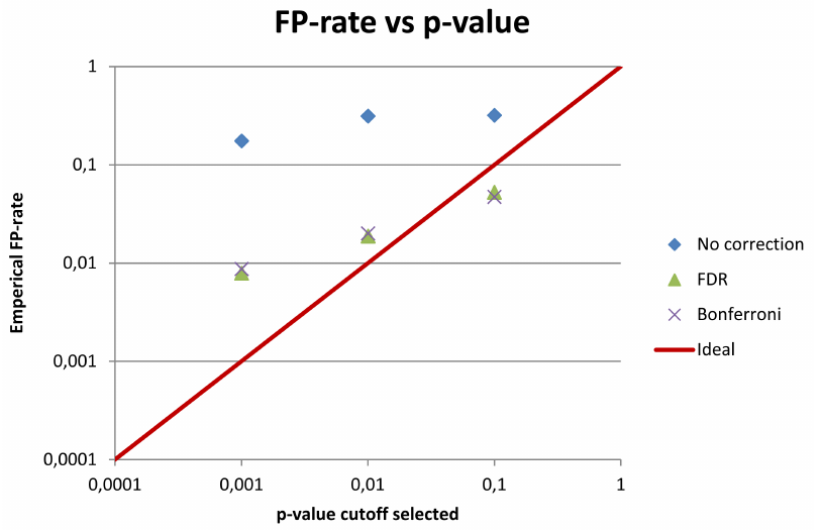


Figure 5.12: Comparison of the Empirical FP-rate for different p-value cutoffs. The blue dots correspond to the approach where no multiple hypothesis correction is used. Both the Bonferroni and the FDR correction lead to an improvement of a factor 10. Achieving an FP-rate below 5% requires limiting the p-value to 10^{-1}

Higher sensitivity than its competitors

In Figure 5.13 three competing algorithms are compared, for their ability to detect multi-species collinearity. The figure shows the number of genes in level-N Multiplicons, with N on the X-axis. In this experiment five vertebrates were compared: human, chimpanzee, mouse, chicken and pufferfish. The difference for the number of genes in level-5 Multiplicons is the most outspoken: Cyntenator has 498 APs, MCScan 416, but i-ADHoRe detects 3296. Note that this result cannot be explained by a higher FP-rate, as has been confirmed in the previous section.

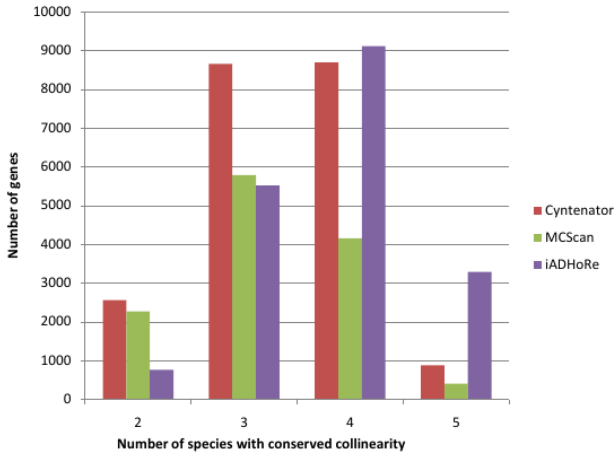


Figure 5.13: Distribution of Multiplicon level for 3 different algorithms for collinearity detection. MCScan is overall the least sensitive of the three algorithms. i-ADHoRe and Cyntenator are similar in the number of detected regions but i-ADHoRe clearly is more capable of detecting multi-species collinearity, as can be seen on the right hand side of the figure.

Does this increased sensitivity also lead to interesting biological results? In the introduction it was already mentioned that i-ADHoRe can process some of the most challenging datasets, such as the Ensembl [10] dataset. The version used in this work (release 57) contains 49 species, with a total of 832,666 genes and 70,161 genomic regions. The total output of the algorithm consists of 237,292 Multiplicons containing 5,204,391 APs.

A typical benchmark for any data mining algorithm, is to verify whether it can reproduce certain, well-known results. This is the case in the Ensembl dataset, where i-ADHoRe finds, as its highest-level Multiplicon, the **HOX-cluster**. This cluster is

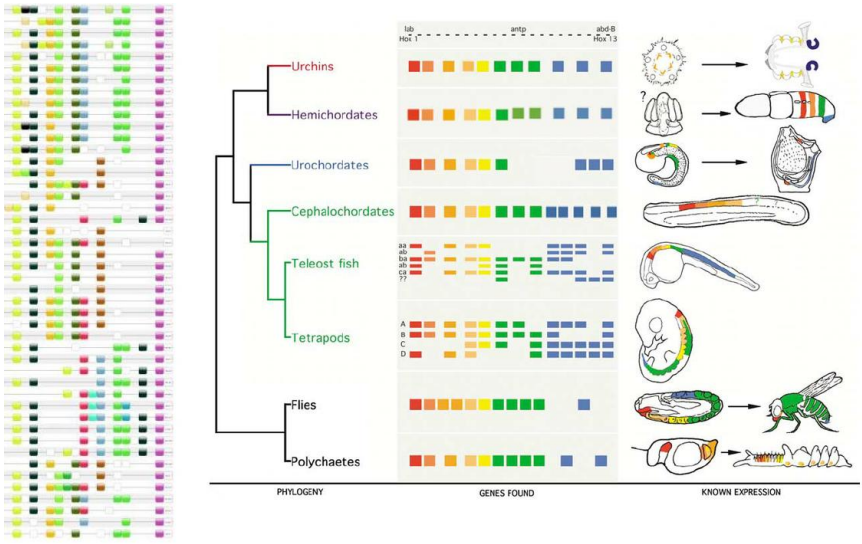


Figure 5.14: The highest-level Multiplicon (level 46) found in the Ensembl Dataset. Left: This Multiplicon corresponds to the well-known HOX-cluster which is associated with the body plan of vertebrates. Alignment Figure from Vandepoele [25]. Right: Phylogenetic tree with the Hox Genes from Swalla [23].

shown in Figure 5.14. The detected cluster corresponds to a profile with 46 collinear regions. The order of genes in this cluster is known to be associated with the body plan in vertebrates.

Better scalability

One of the main features of the i-ADHoRe 3.0 algorithm is its ability to scale in a distributed setting (horizontal scaling).

In the infobox on page 98 two different approaches to scalability are discussed. In terms of *strong scalability* Figure 5.15 examines the speedup of the i-ADHoRe algorithm when the problem size remains fixed but the number of processes goes up.

This is tested on the Stevin HPC infrastructure with the number of nodes going from 1 to 8 and with 8 processes per node. This evaluation again makes use of the Ensembl [10] dataset.

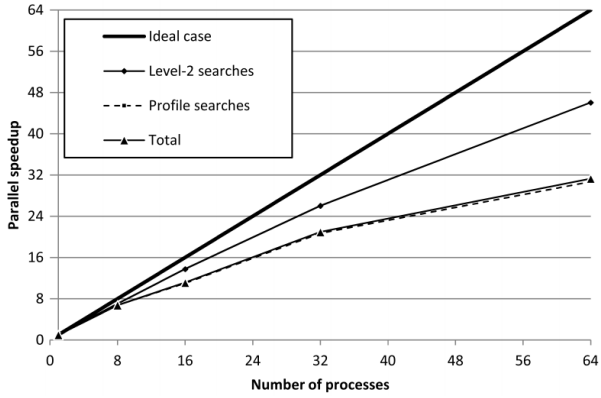


Figure 5.15: Speedup of the i-ADHoRe algorithm for a fixed problem size, the Ensembl dataset, and a growing number of processes. The level-2 algorithm has a higher parallel efficiency than the profile-based search.

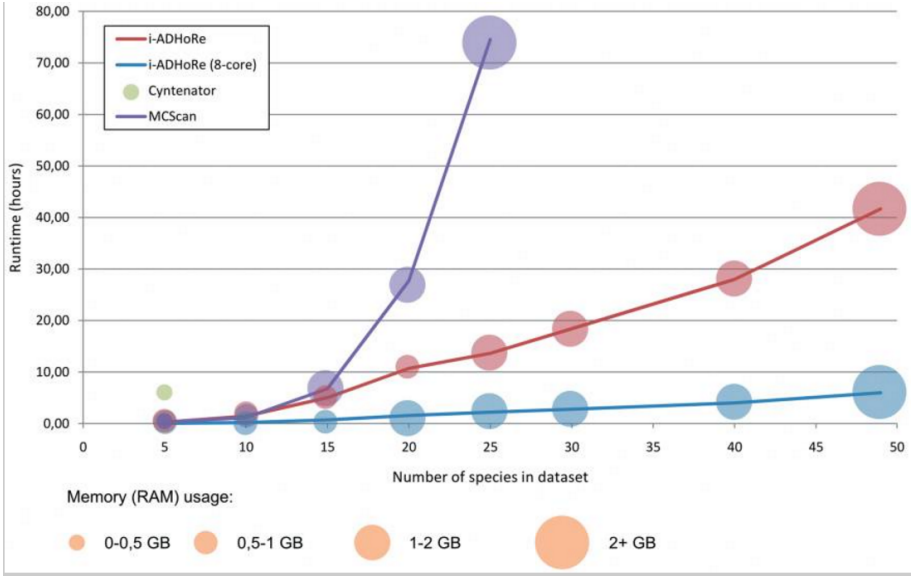


Figure 5.16: Runtime and memory usage comparison of i-ADHoRe and its competitors. Only i-ADHoRe can handle the entire Ensemble dataset and the results with 8 processes indicate that the algorithm can easily handle even larger datasets if the resources grow accordingly.

BACKGROUND: Metrics and Laws of Parallel Performance

The runtime of a parallel algorithm with P processes is T_P. The **Speedup** S_P is then defined as

$$S_P = \frac{T_1}{T_P}$$

There are two important laws that describe the speedup of a system. **Amdahl's law** (strong scaling) describes the inherent upper limit to the speedup if the problem size N is fixed, but the number of processes P goes up:

$$S_P = \frac{sT_1 + (1-s)T_1}{sT_1 + \frac{1-s}{P}T_1} = \frac{1}{s + \frac{1-s}{P}} \xrightarrow{P\to\infty} \frac{1}{s}$$

with s the inherently sequential fraction of T_1 which cannot be sped up.

Gustafson's law (weak scaling) studies the effect of increasing N simultaneously with P. The interpretation is the most straightforward when focusing on **parallel efficiency** $\eta_P = \frac{S_P}{P}$, or the percentage of the optimal speedup which is achieved. The law models the overhead fraction s as a constant for increasing problem sizes N:

$$T_1(N) = T_1(1)\,(s + (1-s)N)$$

With P processes this corresponds to the following runtime:

$$T_P(N) = T_1(1)\left(s + \frac{(1-s)N}{P}\right)$$

Keeping $N = P$ fixed, we finally get:

$$\eta_P(P) = \frac{T_1(P)}{P \cdot T_P(P)} = \frac{T_1(P)}{P \cdot T_1(1)} = \frac{s}{P} + (1-s) \xrightarrow{P\to\infty} 1 - s$$

Gustafson's law thus demonstrates that for a fixed amount of work per node, η_P drops quickly until it converges to $1 - s$.

Figure 5.15 makes a distinction between the two different phases of the i-ADHoRe algorithm. For the level-2 algorithm, which performs a pairwise comparison between all genomic segments, we achieve a speedup $S_{64} = 46$ ($\eta_{64} = 72\%$). As stated earlier, the profile-based search is more difficult to parallelize: the amount of communication is larger, compared to the compute time. This is because the GHMProfiles are much smaller and the collective communications are more frequent. For this phase the achieved speedup is $S_{64} = 32$ ($\eta_{64} = 50\%$).

The number of available genomes is rising quickly. In that light the *weak scaling* of the algorithm is even more important. We compared i-ADHoRe with $P = 1$ and

with $P = 8$ to its competitors in terms of runtime (Y-axis) but also in terms of memory usage (area of dots). The results for an increasing number of species are shown in Figure 5.16. For Cyntenator, only results up to 5 species were obtained. It requires over $T_1^{Cynt}(5) = 6$ hours to analyze five genomes. Even in this small setup MCScan and i-ADHoRe are already an order of magnitude faster, with run-times of $T_1^{MCS}(5) = 19$ minutes and $T_1^{i-AD}(5) = 14$ minutes respectively. In the 8-core setup the full analysis was even further reduced to $T_8^{i-AD}(5) = 3$ minutes.

MCScan cannot handle the entire Ensembl dataset within a reasonable time frame. In a time window of $T_1^{MCS}(20) = 48$ hours 20 species could be processed, but getting to 30 species increases the runtime to $T_1^{MCS}(30) = 168$ hours.

i-ADHoRe on the other hand, even without parallelization can handle the entire Ensembl dataset in only $T_1^{i-AD}(49) = 42$ hours on a single core. With eight cores this runtime can be reduced to approximately $T_8^{i-AD}(49) = 6$ hours. This corresponds to $\eta_8(49) = 88\%$. Using 8 multi-core nodes further reduced the runtime to $T_{64}^{i-AD}(49) = 40$ minutes with $\eta_{64}(49) = 50\%$.

The difference in memory usage can be attributed to i-ADHoRe storing homology in terms of gene families as opposed to MCScan storing gene pairs. The first requires $\mathcal{O}(N)$ storage while the latter requires $\mathcal{O}(N^2)$

5.4 Conclusion

In this chapter we have taken a deep dive in the internal machinery of the i-ADHoRe 3.0 algorithm. Details about the data structures and complexities of the algorithms are provided and more insights into the parallelization approach are given.

The i-ADHoRe is grounded in solid statistical foundations and was implemented to be able to shine in a HPC setting. Both explain why i-ADHoRe 3.0 outperforms the competition of collinear clustering algorithms in terms of sensitivity and scalability. i-ADHoRe detects higher degrees of collinearity and retrieves well-known results such as the HOX-cluster in vertebrates. Furthermore, its efficient implementation make it not only the most performing single-threaded algorithm, but also demonstrates that it can efficiently process the largest datasets, in less than an hour, given sufficient resources. i-ADHoRe 3.0 will remain the state-of-the-art for quite some time to come.

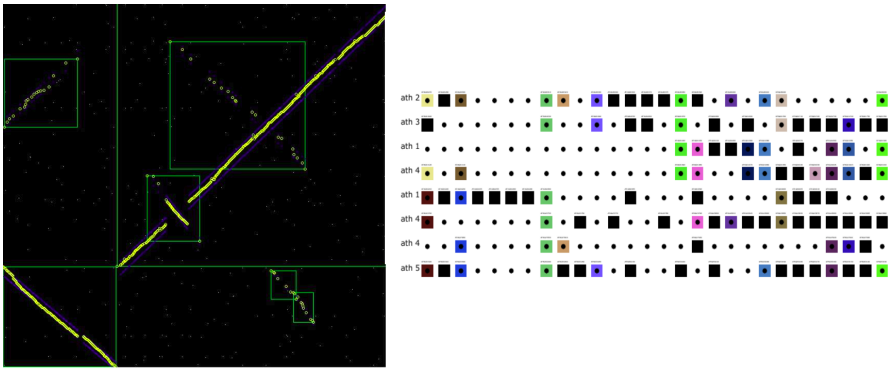


Figure 5.17: iVisualize: A post-processing module was added which allows for visual inspection of the i-ADHoRe results.
Left: An example of a GHM, yellow dots correspond to APs, blue dots correspond to the confidence intervals of the linear fit, the green boxes correspond to the bounding boxes of BaseClusters which survived the quality filters (red if p-value is too high).
Right: An example of level-8 Multiplicon, homologous genes are given the same box color, black boxes are genes with no homologous matches, gaps introduced by the aligner have no box.

An important contribution to guarantee the trustworthiness of the algorithm was an additional module developed for visualizing the algorithm's output. The iVisualize module shows both GHMs and the aligned profiles, as can be seen in Figure 5.17. Inspections revealed multiple issues which would not have been revealed in the absence of visual feedback.

The work on i-ADHoRe 3.0 was well received by the research community which is reflected by its acceptance in a high impact journal (Nucleic Acids Research, impact factor 10.2) and according to Researchgate 93 citations as of June 2018. If we take the ten most cited papers (65-232 citations) from this list, we get an idea about the topics of the citing papers:

- 2 Updates to the Plaza platform of which i-ADHoRe is a component.
- 1 Alternative to the Plaza platform.
- 3 VIB research papers where i-ADHoRe is used to study new genomes.
- 4 non-VIB research papers where where i-ADHoRe is used to study new genomes.

From my point of view it was a very rewarding introduction to fields of algorithm design, computational biology, and parallel computing.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, 1995.

[3] N. H. Bergman. Comparative genomics. Volume 1-2 of Springer Science & Business Media, 2007.

[4] M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, et al. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome research*, 14(4):708–715, 2004.

[5] A. C. Darling, B. Mau, F. R. Blattner, and N. T. Perna. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research*, 14(7):1394–1403, 2004.

[6] C. N. Dewey. Aligning multiple whole genomes with mercator and mavid. *Comparative genomics*, 2008.

[7] C. N. Dewey and L. Pachter. Evolution at the nucleotide level: the problem of multiple whole-genome alignment. *Human Molecular Genetics*, 15(suppl_1):R51–R56, 2006.

[8] S. Dudoit and M. J. van der Laan. Multiple tests of association with biological annotation metadata. *Multiple Testing Procedures with Applications to Genomics*, 2008.

[9] J. Fostier, S. Proost, B. Dhoedt, Y. Saeys, P. Demeester, Y. Van de Peer, and K. Vandepoele. A greedy, graph-based algorithm for the alignment of multiple homologous gene lists. *Bioinformatics*, 27(6):749–756, 2011.

[10] T. Hubbard, D. Andrews, M. Cáccamo, G. Cameron, Y. Chen, M. Clamp, L. Clarke, G. Coates, T. Cox, F. Cunningham, et al. Ensembl 2005. *Nucleic acids research*, 33(suppl_1):D447–D453, 2005.

[11] N. C. Jones and P. Pevzner. An introduction to bioinformatics algorithms. MIT press, 2004.

[12] E. B. Lewis. A gene complex controlling segmentation in drosophila. *Nature*, 276(5688):565–570, 1978.

[13] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[14] S. Proost. Integration of genomic data to study genome evolution in plants. PhD thesis. Ghent University, 2012.

[15] S. Proost, J. Fostier, D. De Witte, B. Dhoedt, P. Demeester, Y. Van de Peer, and K. Vandepoele. I-adhore 3.0—fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic acids research*, 40(2):e11–e11, 2011.

[16] C. Rödelsperger and C. Dieterich. Cyntenator: progressive gene order alignment of 17 vertebrate genomes. *PloS one*, 5(1):e8861, 2010.

[17] B. Rost. Twilight zone of protein sequence alignments. *Protein engineering*, 12(2):85–94, 1999.

[18] C. Simillion, K. Janssens, L. Sterck, and Y. Van de Peer. I-adhore 2.0: an improved tool to detect degenerated genomic homology using genomic profiles. *Bioinformatics*, 24(1):127–128, 2007.

[19] C. Simillion, K. Vandepoele, Y. Saeys, and Y. Van de Peer. Building genomic profiles for uncovering segmental homology in the twilight zone. *Genome research*, 14(6):1095–1106, 2004.

[20] C. Simillion, K. Vandepoele, M. C. Van Montagu, M. Zabeau, and Y. Van de Peer. The hidden duplication past of arabidopsis thaliana. *Proceedings of the National Academy of Sciences*, 99(21): 13627–13632, 2002.

[21] Slowkow. The score matrix for an alignment between gattaca and gcatgcu. 2014. https://commons.wikimedia.org/wiki/File:Needleman-Wunsch_pairwise_sequence_alignment.png

[22] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[23] B. Swalla. Building divergent body plans with similar genetic pathways. *Heredity*, 97(3):235, 2006.

[24] H. Tang, X. Wang, J. E. Bowers, R. Ming, M. Alam, and A. H. Paterson. Unraveling ancient hexaploidy through multiply-aligned angiosperm gene maps. *Genome research*, 18(12):1944–1954, 2008.

[25] K. Vandepoele. Detection of genomic homology in eukaryotic genomes. 2012. https://www.slideshare.net/klaasvandepoele/detection-of-genomic-homology-in-eukaryotic-genomes

[26] K. Vandepoele. Dissecting plant genomes with the plaza 2.5 comparative genomics platform. 2013. https://www.slideshare.net/klaasvandepoele/dissecting-plant-genomes-with-the-plaza-25-comparative-genomics-platform

[27] K. Vandepoele, W. De Vos, J. S. Taylor, A. Meyer, and Y. Van de Peer. Major events in the genome evolution of vertebrates: paranome age and size differ considerably between ray-finned fishes and land vertebrates. *Proceedings of the National Academy of Sciences of the United States of America*, 101(6):1638–1643, 2004.

[28] K. Vandepoele, Y. Saeys, C. Simillion, J. Raes, and Y. Van de Peer. The automatic detection of homologous regions (adhore) and its application to microcolinearity between arabidopsis and rice. *Genome Research*, 12(11):1792–1801, 2002.

[29] K. Vandepoele, C. Simillion, and Y. Van de Peer. Detecting the undetectable: uncovering duplicated segments in arabidopsis by comparison with rice. *Trends in Genetics*, 18(12):606–608, 2002.

Chapter 6

Distributed Comparative Motif Discovery

More is good...all is better.

—The Ferengi Rules of Acquisition nr. 242

Whereas the goal of the previous chapter was to detect sequential patterns in gene sequences, this chapter studies genomes at the DNA sequence level. The pattern discovery specifically focuses on the non-coding promoter sequences. These patterns, so-called *motifs*, are putative binding sites for regulatory proteins which affect the rate of a gene's transcription. The goal of computational *motif discovery* is to generate candidate motifs which can be biologically validated in lab experiments.

The accurate discovery and annotation of regulatory elements is a challenging problem. The growing number of sequenced genomes creates new opportunities for *comparative* approaches to motif discovery: putative binding sites are then considered to be functional, if they are conserved in orthologous promoter sequences of multiple related species. Existing methods for comparative motif discovery usually rely on pregenerated multiple sequence alignments, which are difficult to obtain for more diverged species, such as plants. As a consequence, misaligned regulatory elements often remain undetected.

In this chapter we present a novel algorithm that supports both alignment-free and alignment-based motif discovery in the promoter sequences of related species. Pu-

tative motifs are exhaustively enumerated, as words[1] over the IUPAC alphabet, and screened for conservation using the branch length score. Additionally, a confidence score is established in a genome-wide fashion.

The alignment-free version of the algorithm is more sensitive than the alignment-based and detects *more* motifs while controlling the False Discovery Rate. Furthermore, the exhaustive nature of the algorithm and the lack of heuristics to limit the motif search space, guarantees that *all* motifs are detected and a globally optimal motif is reported. The latter makes the Ferengi quote all the more suitable.

In section 6.1, we give an overview of existing approaches to motif discovery, for which Das [11] is the main inspiration and source for further reading. This section categorizes algorithms according to (i) the motif models used; whether the algorithms are exact or probabilistic; (iii) if alignments are used; (iv) if the relationship between the sequences points to co-regulation or to a common ancestor.

In section 6.2, we discuss Sagot's algorithm for inexact motif discovery which served as the initial inspiration to this work. We proceed with the modifications we made to make it applicable for comparative motif discovery and for a different motif model (IUPAC). The remainder of the section deals with the challenge of translating the local algorithm into a *distributed* motif discovery algorithm. This is accomplished by rewriting the algorithm such that it fits in the MapReduce programming model and can thus be written as a sequence of map() and reduce() operations.

In section 6.3, we review the results generated by applying our *BLSSpeller* algorithm to four monocotyledon plant species. BLSSpeller is a contraction of BLS, which is the name of a conservation metric called Branch Length Score, and Speller which is the name of Sagot's initial motif discovery algorithm. We discuss the algorithmic runtime and memory performance. The validity of the results are assessed by studying the False Discovery Rate. Biological validation comes from observing, that high-scoring motifs are significantly enriched for open chromatin regions in *Oryza sativa* and for transcription factor binding sites inferred through protein-binding microarrays in *Oryza sativa* and *Zea mays*. Furthermore, the method is shown to recover experimentally profiled ga2ox1-like KNOTTED1 (KN1) binding sites in *Zea mays*.

[1] In this chapter we define a 'word' as a character sequence using a certain motif alphabet such as the IUPAC alphabet.

6.1 Introduction to Motif Discovery

Related Work

One of the major challenges in systems biology is gaining a full understanding of gene transcriptional regulation. Transcription factors, for which the binding sites are usually hidden in the promoter sequence of the gene, are in this respect of particular importance. Computational approaches for *de novo* motif discovery can be classified in (a) methods to identify binding sites in promoter sequences of *co-regulated genes* within a single genome and (b) comparative approaches using *homologous sequences* from multiple related species [11].

Co-regulated genes The first category uses clusters of co-expressed genes, which are assumed to be regulated by the same set of transcription factors. A drawback of these methods is that the relationship between co-expression and co-regulation relies on complex regulatory mechanisms, making it *difficult to assemble reliable datasets* since co-expression does not necessarily imply that there is a common binding site involved. Two different algorithmic approaches coexist: the *statistical* [1, 29, 35, 52, 60] (see background) and the exhaustive, *word-based* algorithms. The latter contain *graph-based* approaches [22, 34, 45] and methods based on *index structures* [36, 37, 39].
The graph-based approaches model a set of N gene sequences as an N-partite graph where each node correspond to a k-mer at each position in the sequence. Edges are drawn between k-mers lying within a certain edit distance. The algorithms try to find cliques in this graph.

BACKGROUND: Statistical Approaches to Motif Discovery

Statistical approaches to motif discovery can be subdivided into: adaptations of the EM algorithm [18] and methods relying on Gibbs Sampling [24].
The basic algorithm for motif discovery by Lawrence [32] takes a number of base sequences S and a fixed motif length W and returns a probabilistic model corresponding to the shared motif in these sequences. EM typically learns a set of latent variables, which are in this setting the starting positions $\mathbf{Z}$ of the shared motif in all sequences and a set of base probabilities θ in the motif model, which is a Position Weight Matrix (PWM). Repeat 2-3 until convergence of θ:

1. Initialization: Set all θ probabilities to random values.
2. For each sequence i, compute the position $\mathbf{Z_i}$ ($\mapsto j = 1$) with score I_{model} for the PWM (I_{model} = information content)

$$I_{model} = \sum_{j=1}^{W} \sum_{l \in \Sigma} \theta_{jl} \log(\frac{\theta_{jl}}{\theta_l})$$

3. Given $\mathbf{Z_i}$, estimate the PWM base probabilities (maximum likelihood) θ_{jl} (base alphabet is Σ)

The **Gibbs Sampling** approach is a Markov chain Monte Carlo technique. EM is deterministic, Gibbs sampling is a randomized algorithm.

1. Initialization: Choose a random position $\mathbf{Z_i}$ in every sequence.
2. Randomly leave out one sequence x, and build a PWM with (maximum likelihood) base probabilities θ_{jl} from the other S-1 sites.
3. Given this new PWM, find the optimal position p_{new} in the sequence x that was left out, and set $\mathbf{Z_x} = p_{new}$

Both statistical approaches suffer from local optima. This is typically addressed by re-running the algorithm with different initial values.

Phylogenetic footprinting Due to the growing availability of genome sequences, a second category of algorithms based on **phylogenetic footprinting** emerged [4]: orthologous regulatory regions from multiple species are compared *with the under-*

lying assumption that functional elements evolve at a much slower pace, compared to the non-functional part of the genome, due to selective pressure [3]. Most comparative motif discovery approaches rely in some way on **multiple sequence alignments** (see previous chapter page 74), in which regulatory signals are expected to be well-aligned. Pioneering algorithms in this category are Conreal [3], Phylonet [59], and Phyloscan [8]. More recent algorithms relying on alignments are used to study mosquitoes [46], Fusarium [31], vertebrates [23], and mammals [63].

Misaligned motifs It has, however, been shown that known **regulatory elements are not always correctly aligned** [47], an issue that is further complicated by the different alignments produced by various alignment programs [40].

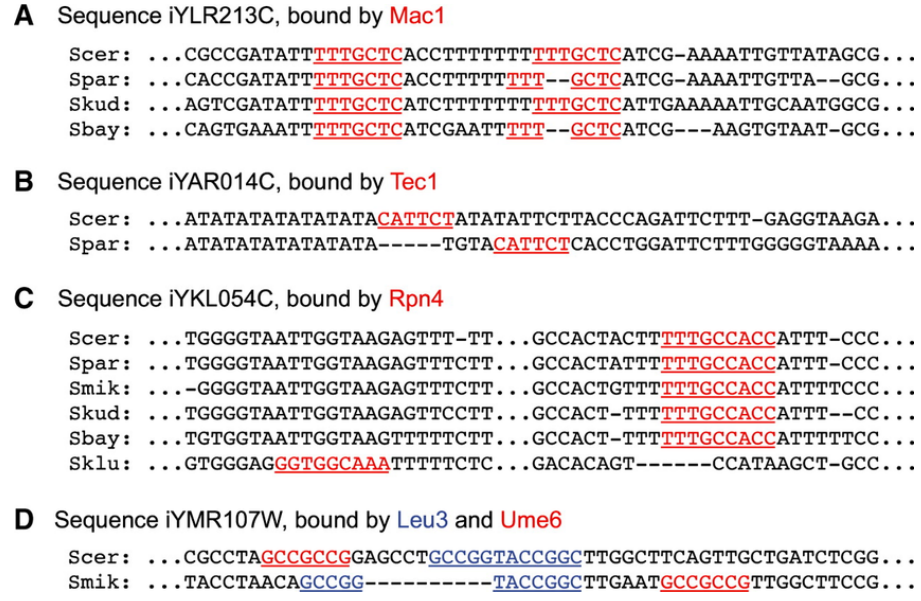


Figure 6.1: 4 examples of well-established Yeast binding sites (red) in orthologous sequences, which are misaligned. In (A) the MSA incorrectly inserts gaps in the binding sites. In (B) the MSA makes an error by aligning non-functional repeats at the cost of aligning the actual binding sites. In (C) the binding site changed orientation, and (D) demonstrates the fact that binding sites can be mobile for example due to binding site turnover.

Transcription Factor (TF) binding sites are *short*, flexible against certain *mutations* and even *mobile*, which explains why they are sometimes misaligned. Mechanisms have been observed that allow the modification of regulatory sequences without altering their function: *divergence driving words* and *binding site turnover*. Regulatory

sequences can diverge freely if the divergence driving words, which are specific short words in the non-coding DNA, are not altered [7]. Since a TF can often bind to multiple similar sites, mutations turning one site into another should not affect regulation. Binding site turnover, on the other hand, is the mechanism where the gain of a redundant binding site allows the loss of a previously functional site [57]. The corresponding TF can then bind to the new site, maintaining the regulatory interaction. This allows binding sites to relocate within the regulatory sequence, making it difficult for alignment algorithms to correctly align them. The problems with using sequence alignments for motif discovery are graphically demonstrated in Figure 6.1 taken from Gordan [26].

Gene families Binding site discovery, especially in plants, has to deal with large divergence times and complex diversification mechanisms such as genome duplications. This makes approaches based on whole genome alignments (WGAs), often used in *de novo* algorithms, impractical. Some of these problems have been addressed in earlier studies. Stark [49] used a mixed approach in a study with 12 Drosophila species, starting from whole genome alignments but allowing for limited motif movement within an alignment. Elemento [20] designed an alignment-free algorithm to discover overrepresented k-mers over the exact ACGT alphabet in pairs of related genomes. Finally, MDOS [62] is a new version of this algorithm with improved statistics.

Benchmarking Motif Models and Algorithms

Computational motif discovery is a research area where many ideas have been tried. It is however difficult to assess which of these ideas works best and what is the accuracy of the predictions w.r.t. biologically validated binding sites.

Benchmarking algorithms Tompa [54] designed a benchmark to compare different algorithms on both artificial and biological datasets. The benchmark consists of both sequences with and without an embedded motif. For ranking the different algorithms a score was devised: the nucleotide correlation coefficient (nCC).

The nCC is the Pearson Correlation coefficient between two binary variables, the first being 1 on all loci covered by a pattern and 0 otherwise. The second binary variable is 1 on all target loci of a motif model. The nCC goes from 1 (= perfect predictions) to 0 for random predictions.

Results for the main algorithmic approaches are shown in Figure 6.2.

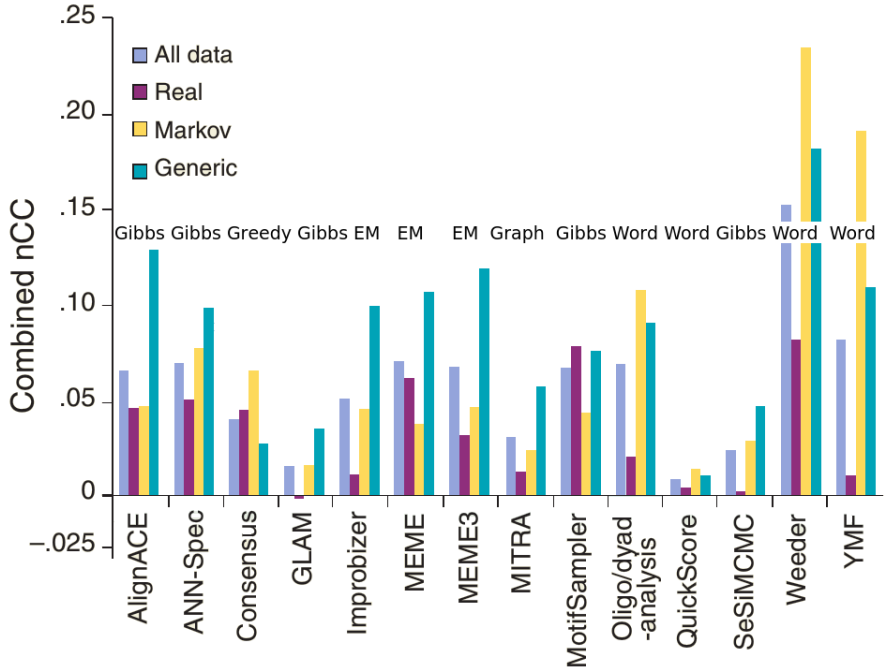


Figure 6.2: Showing the nCC scores for 13 algorithms on 4 different benchmark datasets. Motif algorithm type added, with the highest occurrence of Gibbs Sampling, EM, and Word-based. Weeder, a word-based algorithm with the highest similarity to our algorithm BLSSpeller, has the best performance on all datasets, albeit less outspoken on the 'real' dataset.

Benchmarking motif models Position Weight Matrices [50] (PWM), Mismatch strings [36] (MM), and IUPAC strings [10] are the three most common models to represent motifs. All are defined, together with an example model, in the caption of Figure 6.3. The full 15-character IUPAC alphabet is shown in Table 6.1.

One of the main motivations for designing a new word-based motif discovery algorithm, is the superior performance of Weeder [39]. Weeder shares many ideas with BLSSpeller, but it uses the MM model. Figure 6.4 shows this model has a lower overall performance as compared to IUPAC and PWM models. This figure is taken from Sandve [44], who used the same datasets as Tompa [54], to compare the overall performance of the different models: the IUPAC model scores best, closely followed by the PWM, but the MM string is clearly less accurate.

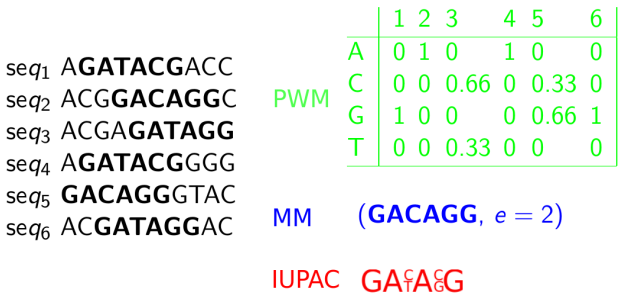


Figure 6.3: For a toy set of 6 sequences, each with a binding site (in bold), three motif models are shown. The *PWM* for a k-character motif is a $k \times 4$ matrix containing the probability for each nucleotide to occur at a certain position. The *MM string* is represented by a DNA string and an integer number e representing the edit distance (Hamming or Levenshtein) between the consensus string and the binding site occurrences. *IUPAC strings* are words written in an extended alphabet containing degenerate symbols.

MM models are probably the least strong in the assessment, since the position of an error in the model is not specified, nor is anything specified about the character substitution. The model therefore is less precise, which makes it more sensitive to false positives.

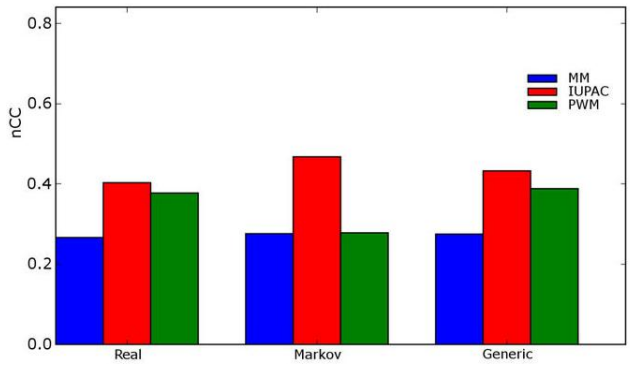


Figure 6.4: Comparing the performance of the different motif models on the Tompa datasets. IUPAC is consistently showing the highest nCC score, PWMs have a comparable performance, but the MM model turns out to be much weaker at representing binding sites.

A weakness of the 3 models described so far is the lack of support for inter-positional dependence.

> **Example:** Assume that we have a binding site where position 1 is A and position 3 is C, or position 1 is T and position 3 is G. In the PWM this motif will have 0.5 probabilities at positions 1 and 3, for IUPAC strings there will be a two-fold degenerate character at both positions, and for the MM model $e = 2$.

A possible remedy could be to cluster the motifs after running a motif discovery algorithm. The previously described motif might actually show up as 2 different IUPAC motifs of which one union consensus motif can be created.

BLSSpeller: Approach

The BLSSpeller algorithm distinguishes itself along multiple dimensions:

Phylogenetic Footprinting In this chapter, four closely related monocotyledonous plant species are studied using a phylogenetic footprinting approach: *Oryza sativa ssp. indica* (osa), *Brachypodium distachyon* (bdi), *Sorghum bicolor* (sbi), and *Zea mays* (zma). The DNA sequences in a gene family are thus evolutionary related, i.e. we assume they have a common ancestral sequence.

Gene-Centric We adopt a gene-centric approach, where the promoter sequences of orthologous genes are grouped into *gene families* (as opposed to Whole-Genome Alignment (WGA)).

Word-based Discovery Algorithm A word-based discovery algorithm was designed to exhaustively report all *genome-wide conserved* motifs. The term *conserved* relates to the occurrence of the motif in multiple promoter sequences of a particular gene family. *Genome-wide conservation* relates to the fact that this conservation occurs in more gene families than what is expected by chance.

IUPAC Motif Model Motifs are modeled as words (k-mers) over an alphabet that contains the 4 bases (ACGT) and (optionally) additional degenerate characters from the IUPAC alphabet [10]. This degeneracy allows a motif to model a collection of binding sites. The specifics of the IUPAC alphabet are given in Table 6.1.

Alignment-free The algorithm can be run in both *alignment-free* or *alignment-based* mode. In case of alignment-free discovery, the conservation of a motif is scored irrespective of its orientation or position within a promoter sequence. This relaxed

Table 6.1: IUPAC degenerate alphabet for DNA.

Symbol	Meaning	Origin of designation
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
R	A or G	puRine
Y	T or C	pYrimidine
M	A or C	aMino
K	G or T	Ketone
S	C or G	Strong interaction
W	A or T	Weak interaction
H	A or C or T	not-G
B	C or G or T	not-A
V	A or C or G	not-T
D	A or G or T	not-C
N	A or C or G or T	

definition of conservation was previously used by Gordan [26] and is especially relevant when studying more diverged species for which accurate multiple sequence alignments are difficult to generate. Alignment-based discovery adds the constraint that motifs must be aligned, i.e., occur at the same position in the multiple sequence alignment.

BLS method supports partial conservation Robust algorithms for comparative genomics are expected to gain in power when more related species are added. Most studies so far only consider motifs that are conserved within *all* organisms. The Branch Length Score (BLS) was developed to quantify motif conservation in a biologically meaningful manner and ranges from 0% (not conserved) to 100% (conserved in all sequences). The BLS takes the phylogenetic relationships between the species into account by representing a relative evolutionary distance over which a candidate binding site is conserved within a gene family. The BLS was first used in a comparative study with 12 Drosophila genomes [49] and allows studying motifs only conserved in subsets of the organisms.

Exhaustive algorithm Whereas most word-based algorithms avoid exploring the full motif space by using greedy approaches, our method is unique in the sense that it is exhaustive. MDOS [62] only processes promising k-mers and gradually adds degeneracy if this improves the conservation score. Kellis [30] and Stark [49] use

the mini-motifs approach [28] only processing promising trinucleotide duos before adding degeneracy. Here, every word is considered by the discovery algorithm. The only restrictions applied are on the motif length $k \in [k_{min}, k_{max}]$ and the maximal number of degenerate characters e_{max}. The advantage of this approach is that it avoids issues related to local optima.

Parallelization based on MapReduce In order to strongly reduce the runtime and avoid excessive memory requirements, the MapReduce programming model [17] was adopted as a means to take advantage of a parallel, distributed-memory cloud computing environment. By enabling disk I/O to store intermediate results, the current MapReduce implementation overcomes the memory bottleneck in a prototype implementation of this software that relied on the Message Passing Interface (MPI) for parallelization [12].

Personal Contributions and Acknowledgements

The ideas behind the BLSSpeller algorithm are a combined effort of Prof. Fostier, Prof. Dhoedt, Prof. Vandepoele, and myself. The full implementation is my personal work. Dr. Decap ran a number of performance benchmarks on the Ugent HPC as requested by the journal reviewers. Dr. Van de Velde en Dr. Van Bel contributed to the biological validation of the algorithm. The contributions of Dr. Van de Velde in terms of writing the paper mainly correspond the section on correlating the BLSSpeller predictions with OCR regions. The other parts of the paper are written by Jan Fostier and me.

This work was presented on a number of international bioinformatics conferences: ISMB [13], ECCB [14], and Benelux Bioinformatics [16]. The full results of this work are published in two research papers: a paper about the MPI implementation at PPAM [12] and a paper about the MapReduce approach published in the Bioinformatics journal [15].

We acknowledge the support of Ghent University (Multidisciplinary Research Partnership 'Bioinformatics: From Nucleotides to Networks') and HPC staff for technical assistance. Part of the computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, the Hercules Foundation and the Flemish Government department EWI.

6.2 BLSSpeller: Distributed Motif Discovery

Sagot's Speller Algorithm

In the work by Sagot [36], which serves as the initial inspiration for the BLSSpeller algorithm, one tries to find a solution to the following problem:

The Common Motifs Problem

Given a set of N sequences s_i (for $1 \leq i \leq N$) and two integers $e \geq 0$ and $2 \leq q \leq N$, find all models m such that m is present in at least q distinct sequences of the set with no more than e mismatches.

A naive solution to find all exact patterns ($e = 0$) with lengths $k_{min} \leq k \leq k_{max}$ consists of enumerating all possible patterns over an alphabet Σ (ACGT). For each pattern the *quorum* (= number of distinct sequences it matches) constraint is checked ($q \geq 2$). In its most naive form this algorithm tests all patterns of a certain length (|{k-mers}| = Σ^k) and scans all sequences one position at a time, this requires $\mathcal{O}(4^k \cdot k \cdot S_{tot})$ character comparisons, with S_{tot} the sum of all sequence lengths.

This can be *sped up* by making use of a *Generalized Suffix Tree* (GST). This is an *index structure* which can be built in $\mathcal{O}(S_{tot})$ time and space complexity and allows the lookup of an exact pattern with length k in only $\mathcal{O}(k)$ time, thus independent of S_{tot}! We will elaborate on this data structure in the next section. Using this data structure leads to an improved runtime complexity of $\mathcal{O}(S_{tot} + \Sigma^k \cdot k)$: the first term corresponds to the building time of the GST, the second term to the pattern lookups.

The factor Σ^k can be dramatically decreased for patterns with increasing lengths: if $\Sigma^k \geq S_{tot}$. The intuition behind this is that there are only S_{tot} possible starting positions for the pattern in the dataset. This calls for a new algorithm which is *sequence-driven* instead of *pattern-driven*. This means that the finite length of the sequences can be used to limit the pattern search space or, differently put, that only patterns should be evaluated which occur in the text! The complexity of this *sequence-driven algorithm* should therefore be: $\mathcal{O}(S_{tot} + \min(\Sigma^k, S_{tot}) \cdot k)$.

This gives rise to the following branch-and-bound condition(s):

Branch and Bound conditions

1. All patterns for which a prefix p[1..l] does not occur in the sequences can be discarded as they will also have no occurrences.
2. Even stronger: All patterns p for which a prefix p[1..l] does not meet the quorum constraint can be discarded as they will also not meet this constraint.

A simplified version of the recursive Speller algorithm is shown on the next page. Internally, the GST is used to check if a pattern has matches and what is its quorum. This requires only $\mathcal{O}(1)$ operations. This algorithm can be trivially modified to deal with patterns of varying length. In fact, an upper boundary for the pattern length is not required, as the quorum constraint also serves as a stop condition in the recursion.

Exact Motif Discovery Algorithm

```
spellModels(model, pos, k, q_min):
      if length(model) == k:
            store(model)
      else:
            for c in Alphabet(ACGT):
                  model_new = model + c
                  pos_new, q_new = GST.move_fromto(pos, c)
                  if pos_new != -1 AND q_new >= q_min:
                        spellModels(model_new, pos_new, k, q_min)
```

To turn the above algorithm into an *inexact discovery algorithm* following modifications have to be made:

- A model is now defined as a pattern and an integer e, which represents the number of allowed *mismatches*.
- A model with mismatches is the union of multiple exact patterns, therefore the data type of the variable `pos`, which corresponds to the matching positions in the tree, is now an array of tuples instead of a single matching position. The tuples take the following shape: (x, x_{err}) Hereby x is a single locus in the GST and x_{err} the number of errors needed to get there given an exact pattern `model`.

- The `move_fromto` function of the suffix tree therefore also needs to check multiple starting tuples and generates a new set of (x', x'_{err}) tuples for the extended pattern `model_new`.

  Example: Suppose we have a pattern ('ACT', $e = 1$) matching at $(x, 0)$,$(y, 0)$, and $(z, 1)$. We will try to extend this pattern with all characters in the DNA alphabet Σ. For every possible extension the `move_fromto` function now has to iterate over sites x, y and z to verify whether the new pattern has any matches and if so what is the associated quorum. For `model_new` = 'ACTA', suppose x is at an internal node of the GST with two outgoing edges starting with 'A' and 'C' (positions x_A and x_C). This will result in a match $(x_A, e = 0)$ and an inexact match $(x_C, e = 1)$. Matches which violate the condition $e \leq e_{max}$ are discarded.

The size of the *search space* for all MM model patterns with lengths from $k_{min} \ldots k_{max}$ and e_{max} errors can be calculated as:

$$|patterns| = \sum_{e=0}^{e_{max}} \sum_{k=k_{min}}^{k_{max}} \Sigma^k \tag{6.1}$$

Sagot calculates the *complexity* of the inexact discovery algorithm for a single k and e as $\mathcal{O}(S_{tot} + k \cdot \min(\mathcal{V}(e,k) \cdot S_{tot}, \Sigma^k))$. In this formula $\mathcal{V}$ is the maximum size of the *pattern neighborhood*. This is the amount of patterns which can match with a single position in a sequence: For $e = 0$ only a single motif can match at a certain position, for $e = 1$ there can be a mismatch at k positions in the motif and each time Σ characters can be at that position. For a general e this corresponds to $\binom{k}{e}\Sigma^e \propto (k\Sigma)^e$.

Generalized Suffix Trees

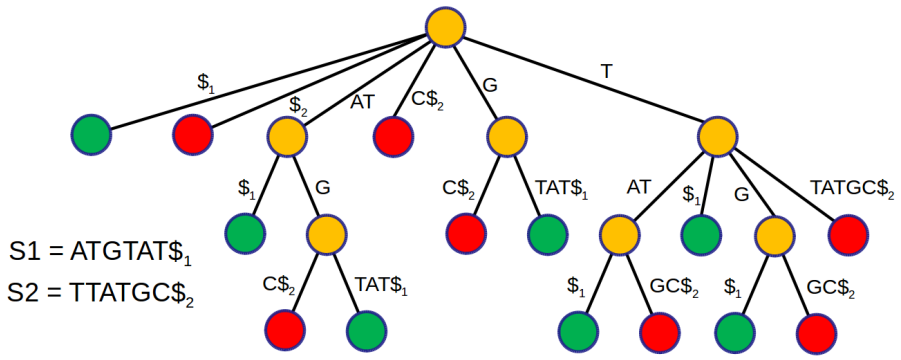


Figure 6.5: Generalized Suffix Tree for two DNA sequences. Internal nodes are shown in orange, leaf nodes corresponding to S1 are in green, leaf nodes corresponding to S2 are in red.

So far the GST has been treated as a black box. We'll use the reference work by Gusfield [27] to provide some formal definitions:

BACKGROUND: Suffix Tree Definition

1. A Suffix Tree T for an m-character string S is a rooted directed tree with exactly m leaves numbered 1 to m.
2. Each internal node, other than the root, has at least two children and each edge is labeled with a non-empty substring of S.
3. No two edges out of a node can have edge-labels beginning with the same character.
4. The key feature of a Suffix Tree is that for any leaf i, the concatenation of edge labels on the path from root to leaf i exactly spells out the suffix of S that starts at position i, S[i..m].

A Suffix Tree is an index structure corresponding to a single sequence. A *Generalized Suffix Tree* (GST) is a suffix tree for a set of sequences. The properties over the Suffix Tree can be satisfied by appending a unique termination symbol per sequence and concatenating the individual strings. If we discard substrings crossing the termination symbols, we get a tree similar to the example in Figure 6.5.

All internal nodes correspond to repeated patterns. The two longest common substrings can be read by concatenating the path from the root of the GST to the internal nodes with the biggest string depth: ATG (AT+G) and TAT (T+AT).

For exact string matching the common motifs problem therefore corresponds to identifying the internal nodes in the GST for which at least two different termination symbols are in the subtree, the number of different '$' symbols is the quorum of a pattern! The algorithm itself encounters these internal nodes during a depth-first traversal of the GST, which corresponds with a depth-first traversal of the pattern space, albeit it while meeting the previously defined branch-and-bound conditions.

Bitvectors As it currently stands, calculating the quorum is an expensive operation. It is however possible to decorate the internal nodes of the GST with bitvectors. As can be seen in Figure 6.6, the bitvectors correspond to the unique sequence terminators in the subtree. The decoration of the internal nodes is a simple bottom-up traversal of the GST. The bitvector of every internal node is then calculated as a

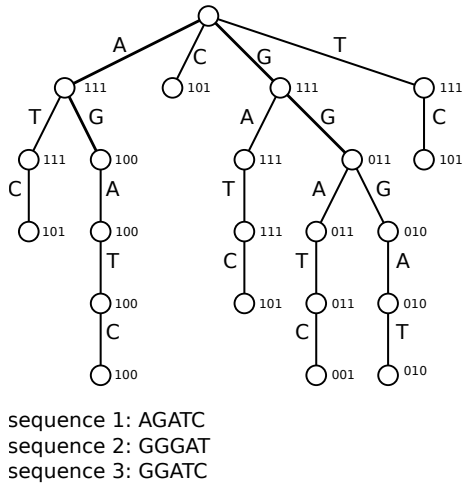


Figure 6.6: GST corresponding to 3 DNA sequences. To simplify the figure, termination symbols are omitted and a trie is depicted (one character per edge)

bitwise OR operation of the bitvectors of its children. The decoration algorithm can be run in $\mathcal{O}(S_{tot})$ time complexity.

Construction So far, we have only mentioned that a GST can be constructed in $\mathcal{O}(S_{tot})$ time and space complexity. Linear construction time algorithms have been devised by Mccreight [38] and Ukkonen [55]. These algorithms also generate *suffix links* between internal nodes, but these are not required in motif discovery algorithms.

Giegerich [25] devised a lazy top-down algorithm for suffix tree construction, which turns out to be faster for most use cases in practice, although the complexity is $\mathcal{O}(S_{tot} \log(S_{tot}))$. This algorithm is used in the implementation of the GSTs in this work. The construction algorithm bears a lot of similarity with a simple radix sort algorithm[2]. In the context of GSTs for motif discovery this approach is efficient as the GST doesn't have to be built entirely: we are only interested in a limited string depth k_{max}. In this context the construction algorithm is linear: $\mathcal{O}(k_{max} \cdot S_{tot})$.

To get a complete overview of the field of index structures in the context of genome analysis we refer to the work of Ghent University colleague Michael Vyverman [58].

[2] https://en.wikipedia.org/wiki/Radix_sort

Motif Discovery in Orthologous Plant Sequences

The BLSSpeller algorithm, contrary to Sagot's speller algorithm, deals with orthologous promoter sequences (as opposed to co-regulated sequences).

The dataset contains orthologous sequences of 4 different monocot plant species:

- *Oryza sativa ssp. indica* (osa) is better known as Asian Rice. The *indica* variant of rice is mainly bred in tropical regions in Asia and grows submerged. The *japonica* is mainly cultivated in dry regions.
- *Brachypodium distachyon* (bdi) mainly serves as a model organism when studying grass species but is of currently not of agricultural interest.
- *Sorghum bicolor* (sbi) is an edible grass species (Dutch: 'kafferkoren') which is mainly used in fodder.
- *Zea mays* (zma) is the standard mays, also called corn, and is one of the primary products in European agriculture.

Based on conserved gene content and genome organization, these grass species are considered to be a single genetic system [2], making a comparative motif discovery approach feasible.

The orthology was inferred using the 'integrative orthology viewer' in the PLAZA 2.5 platform [42, 56]. the PLAZA platform itself internally makes use of:

- BLAST: A local alignment algorithm introduced on page 76.
- TribeMCL [21]: for protein family clustering.
- OrthoMCL [33]: for the identification of orthologous groups.
- FastTree [41] and Notung [9]: for inferring phylogenetic trees from alignments.

BACKGROUND: Types of Homology

Speciation and gene duplication are two mechanisms which can lead to different versions of a particular gene:

1. **Homologs:** All genes that share a common ancestor are said to be homologous.
2. **Orthologs:** Homologous genes which reside in a different organism and have thus evolved separately are said to be orthologous.
3. **Paralogs:** Homologous genes which are the result of a gene duplication are said to be paralogous. The genome thus has two copies of the same gene.

Things get more complicated if there is a combination of duplication and speciation. This is further elaborated in Figure 6.7 (from CoGepedia[a]):

For this case a naming convention was suggested by Sonnhammer [48],

1. **In-Paralogs:** In-paralogs are paralogs which reside in the same organism.
2. **Out-Paralogs:** Out-paralogs are two different versions of a duplicated gene which have evolved further after speciation.
3. **Orthologs:** The same version of the duplicated gene but in a different organism, which have thus independently evolved.

[a] https://genomevolution.org/wiki/images/5/53/Otu.png

Homologous (i.e. orthologous and paralogous) genes were grouped in gene families and their promoter sequences 2 kbp upstream from the translation start site were extracted. In its most simple form, a family consists of four orthologous genes: one from each organism. In that case, the phylogenetic tree by Reinke [43], shown in Figure 6.8, is used.

For gene families that comprise one or more paralogs, gene family-specific phylogenetic trees can be constructed that take into account the specific order in which the duplications and speciation events occurred. For simplicity, we assume that all paralogous gene duplications occurred recently. This is modeled by adding a bifurcation with a branch length of zero to the phylogenetic tree which means that

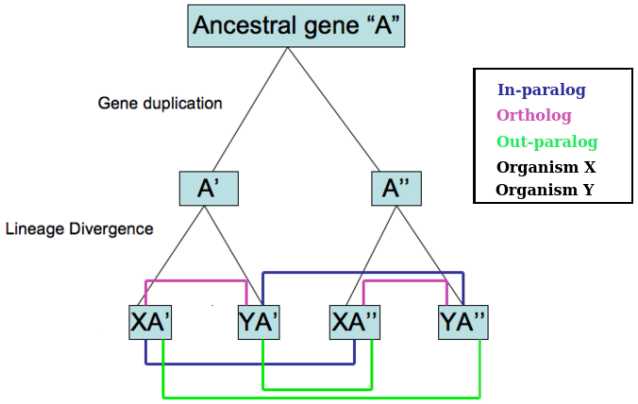


Figure 6.7: Different types of paralogy. This figure demonstrates a hypothetical situation where first a gene A was duplicated, resulting in a gene A' and A", these genes are called *paralogs*. Next the ancestral organisms speciated into an organism X and Y. If we stay within a single organism these genes are now called *in-paralogs*. If we stick to the same gene duplicate, for example A', then XA' and YA' are called orthologs. The weakest relationship is between A' and A" but in different organisms, these are called *out-paralogs*.

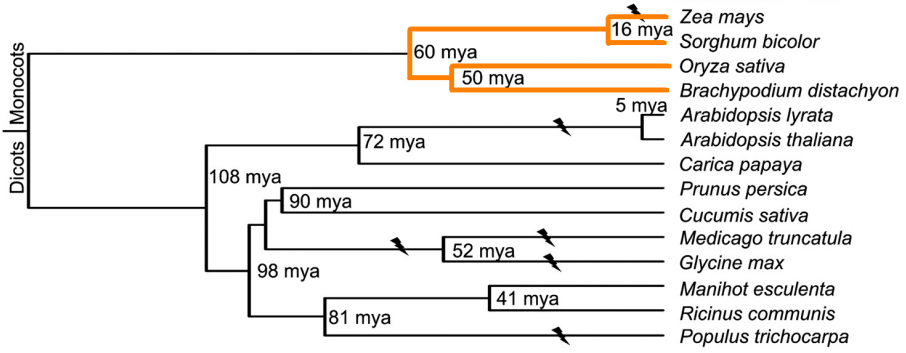


Figure 6.8: Reineke estimated the divergence times (mya) between different monocot and dicot species. Highlighted are the 4 monocot species used in the BLSSpeller dataset. Lightning symbols correspond to whole-genome duplication events.

only conservation between different species contributes to the branch length score. Note that besides promoter regions, additional homologous sequences of interest (e.g., intronic regions) could be added to the input dataset.

The transformation in going from whole genome sequences to gene families is visually depicted in Figure 6.9

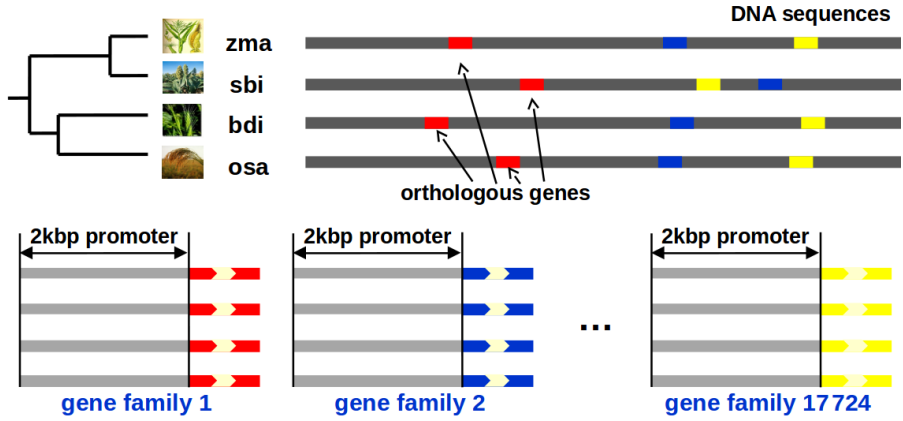


Figure 6.9: Starting from the full genome sequences of the 4 monocot species, orthologous genes are identified and grouped into gene families (17,224 in total). For every gene family we analyze the associated 2kbp promoter sequences for common motifs.

BLSSpeller

Main Differences with Sagot

The BLSSpeller motif discovery algorithm has a lot in common with the speller algorithm of Sagot. The pseudo-code for the exact motif discovery algorithm on page 117 remains roughly valid, apart from the following details.

- The **alphabet** used to enumerate all possible patterns is now the IUPAC alphabet. In most of the simulations we use a restricted version of this alphabet with only 11 characters, eliminating the 3-fold degenerate characters.
- The **quorum** is replaced by a conservation metric (**BLS**) which is a function of the phylogenetic tree.
- The **index structure** has to be able to deal with degenerate symbols.

- Sagot's algorithm focuses on a **single dataset** of sequences. BLSSpeller deals with a **large collection** of gene families. For each gene family the motif discovery algorithm is run, but afterwards these results are aggregated
- The sequences which are compared are evolutionary related (gene family). This makes it more difficult to separate a true pattern from a pattern which is conserved by chance as the sequences are more similar, also in their nonfunctional parts. This is counteracted by employing **genome-wide conservation metrics**.
- Having to process multiple independent datasets at once opens new opportunities for **parallelization** which are tackled using the MapReduce framework.

The input of BLSSpeller consists of gene families containing homologous promoter sequences from related species. The algorithm consists of an *intrafamily* and an *interfamily* step with a sorting step in between. The intrafamily step is where the partial overlap with Sagot's algorithm occurs. The interfamily step deals with genome-wide conservation and has no counterpart in Sagot's algorithm.

Intrafamily step: Conservation within a gene family

For all gene families individually, all words with a length between $k_{\min}$ and $k_{\max}$ characters that occur in any of the sequences, are exhaustively enumerated and their degree of conservation within that family is quantified. Words are spelled in the IUPAC alphabet or a subset thereof. Up to $e_{\max}$ degenerate (i.e., non-ACGT) characters are allowed per word. The intrafamily phase can operate in alignment-free or alignment-based mode.

AF mode In the *alignment-free approach*, a generalized suffix tree (GST) is constructed [25] from the promoter sequences and their reverse complements in the gene family. To enumerate all words that exist in the sequences, we rely on a modified version of the algorithm by Sagot [36], similar to what is used in the Mosdi tool [37]. A depth-first traversal of the GST is performed, examining a single word during each step.

An example of a IUPAC pattern matching in a GST is shown in Figure 6.10. For simplicity, the reverse complement of the sequences is not represented in this example. Each substring contained by any of the sequences is 'spelled out' along a path that

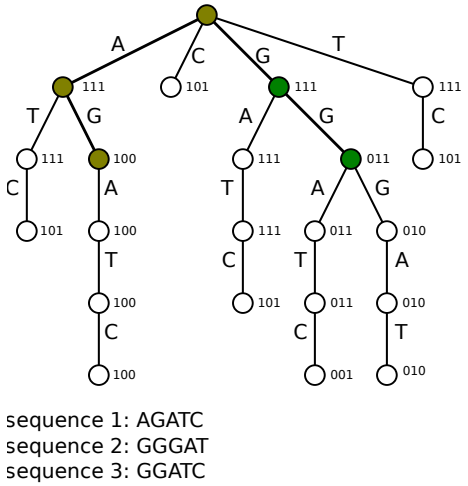


Figure 6.10: Retake of Figure 6.6, here we want to highlight matching of degenerate IUPAC pattern in the GST data structure. The green dots correspond to the matching paths the IUPAC pattern "RG". This also demonstrates the higher precision of IUPAC model as compared to MM strings: the MM string for the model (AG, e=1) would also match with the path 'AT'.

originates from the root of the tree. A bitvector at each node of the GST represents the sequences that contain the substring implied by the path from the root to that node, e.g., '101' denotes occurrences in sequences 1 and 3 but not in sequence 2.

To get a high-level understanding of the modified speller algorithm, we'll focus on Figure 6.10 in the following example:

> **Example:** We assume that we are currently processing the word RG (with R being the IUPAC character representing A or G). The algorithm will then have tracked paths AG and GG in the GST (green-colored nodes in Figure 6.10). By taking the bitwise OR operation of the bitvectors contained by the respective nodes, i.e., '100' for AG and '011' for GG, it is immediately established that RG is contained by all three sequences. In later steps, longer words that have RG as a prefix are considered. Words of length 3 that will be considered are the words RGA, RGG, RGR and RGN. Note that e.g. RGC is never considered as no such path exists in the GST. Consequently, words that contain RGC as a prefix are also never considered. This branch-and-bound condition significantly reduces the number of words to be considered compared to exhaustively scoring all possible existing words over the IUPAC alphabet.

AB mode The *alignment-based mode* requires a pregenerated Multiple Sequence Alignment (MSA) of the orthologous promoters in a gene family. Dialign-TX [51] was chosen to create these MSAs in view of good results on a non-coding alignment

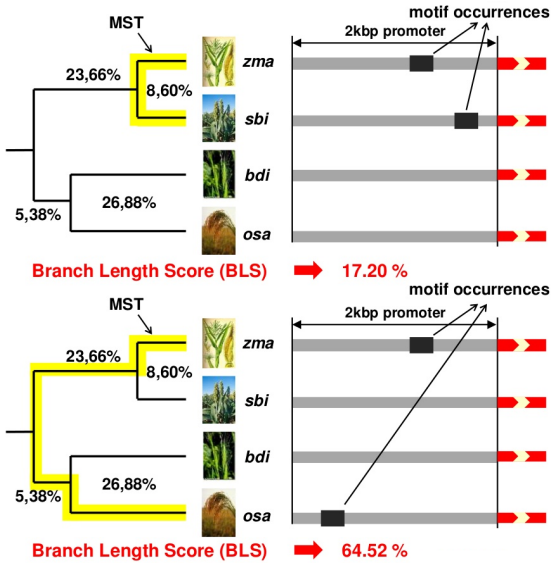


Figure 6.11: Demonstration of how motif BLS scores are calculated. A motif's bitvector corresponds to the different genes which have a motif occurrence. These occurrences then correspond to the leafs of the phylogenetic tree on the left. Next, the MST (yellow) containing these leafs is calculated. The BLS is then the sum of the normalized (sum is 100%) edge lengths in this MST.

benchmark [40]. For every position in the alignment, a small GST is generated containing only the suffixes of the sequences that start at that position. The same Speller algorithm is run to report all words and the sequences in which they occur at aligned positions, again using the IUPAC alphabet.

BLS conservation metric Instead of using the quorum as a *conservation measure*, the degree of conservation in each gene family is quantified using the *branch length score (BLS)*. Given the sequences in which the word occurs, the BLS can be calculated by finding the minimum spanning tree (MST) that connects these sequences in the phylogenetic tree. The sum of the weights of the horizontal branches in the minimum spanning tree then represents the BLS [49]. The branch lengths are calculated from Reineke's tree in Figure 6.8, by normalizing the lengths, such that they add up to 100%. A demonstration how the BLS is calculated is given in Figure 6.11.

In *alignment-based mode*, the same motif can occur at multiple aligned positions within a single family; in that case only the *highest BLS value* is used. Only words

for which the BLS exceeds a prespecified threshold T are retained. Such words are said to be *conserved* within the gene family.

In practice the repeatedly performing the *MST calculation can be avoided*. The number of motifs per gene family can be huge, while the size of the phylogenetic tree is quite small. This can be exploited in the BLS calculations. Instead of calculating a BLS score for every motif from its bitvector we observe that there are only 2^S possible BLS scores, with S the number of genes in a particular gene family. It is therefore beneficial to pre-compute all species combinations and store them in a lookup table.

IUPAC search space The number of patterns considered in IUPAC motif discovery is much larger than in the case of MM strings, as described with formula 6.1. An upper bound to the number of motifs considered in case of IUPAC models is given by:

$$|patterns| = \sum_{k=k_{\min}}^{k_{\max}} \sum_{e=0}^{e_{\max}} \binom{k}{e} \Sigma^{k-e} d^e \tag{6.2}$$

Here d is the number of degenerate characters in the alphabet, i.e., $d = 1$ when using the ACGT+N alphabet, $d = 7$ when using the restricted IUPAC alphabet ACGT+RYSWKM+N and $d = 11$ for the full IUPAC alphabet. The factor d^e is the reason IUPAC motif discovery is much more challenging than MM motif discovery.

In Sagot's algorithm the quorum was used as *branch-and-bound* criterion. The BLS-score closely resembles this quorum (also monotonously decreasing with an increasing pattern length) and can thus be used in the same fashion. If a certain word is found to be conserved with a BLS lower than the lowest *BLS threshold* used. In that case, the word does not need to be extended as such 'extended' words will have at most an equal BLS.

> **Example:** the word AG in Figure 6.10, occurs only in sequence 1 (BLS = 0%). Therefore, there is no need to visit e.g. AGA or AGAT.

Note that the GST can be truncated at a depth of $k_{\max}$ and contains only ACGT characters.

Interfamily Step: Genome-wide conservation

Conserved Family Count The conserved words of all gene families are sorted according to base content and partitioned into *permutation groups* whose elements

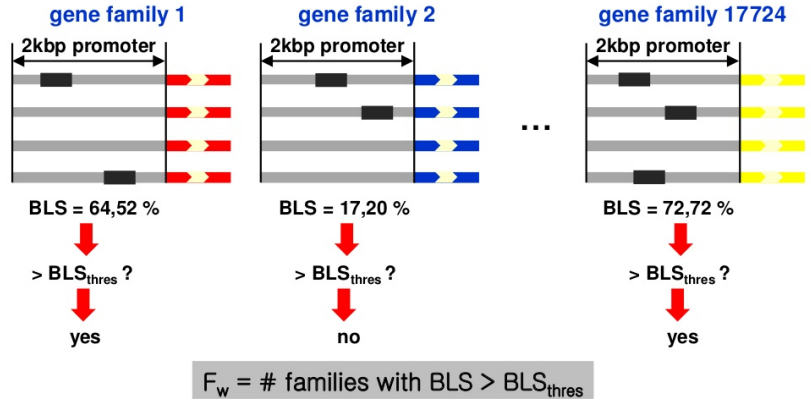


Figure 6.12: Visual explanation of the function F(T), or the amount of gene families for which a pattern satisfies a BLS threshold T, or the *conserved family count*.

are permutations of each other. All words in a permutation group hence have the same length, base content and degeneracy.

Example: The words AWTC, WTAC and CAWT belong to the same permutation group.

The number of occurrences for each distinct word within a permutation group is counted. This number corresponds to the number of gene families in which that word is conserved with a BLS $\geq T$ and is referred to as the *conserved family count* $F(T)$.

Genome-wide conserved motifs are selected based on the fact that they have a conserved family count $F(T)$ that is (much) higher than the median conserved family count of the member instances of their permutation group. This median value, denoted as $F_{\text{bg}}(T)$ (bg = background) represents the *expected* conserved family count for a word in that permutation group. $F_{\text{bg}}(T)$ is approximated by randomly generating a large number (default=1000) of instances of the permutation group and computing the median value for the conserved family count. Note that some of those random instances can have a conserved family count equal to zero.

Confidence score A confidence score C, adopted from Stark [49], is obtained for each word in the permutation group by comparing $F(T)$ and $F_{\text{bg}}(T)$ as follows:

$$C(T) = 1 - \frac{F_{\text{bg}}(T)}{F(T)} \tag{6.3}$$

Words for which $F(T) \geq F_{\text{thres}}$ and $C(T) \geq C_{\text{thres}}$ are considered *genome-wide conserved* motifs and are retained by the method where F_{thres} and C_{thres} denote user-defined thresholds. The output of the method consists of an exhaustive list of motifs which satisfy these thresholds, along with the $F(T)$ and $C(T)$ metrics.

Multiple thresholds in a single run Similar to Stark [49], rather than using a single threshold T, multiple BLS thresholds T_i can be used in a single run. The confidence score $C(T_i)$ is then computed for all thresholds T_i individually, i.e.

$$C(T_i) = 1 - \frac{F_{\text{bg}}(T_i)}{F(T_i)} \tag{6.4}$$

Here, $F(T_i)$ denotes the number of families in which the motif is conserved with a BLS higher than the threshold T_i. Similarly, $F_{\text{bg}}(T_i)$ is the corresponding value for the background model. Words for which $F(T_i) \geq F_{\text{thres}}$ and $C(T_i) \geq C_{\text{thres}}$ for *any* of the BLS thresholds T_i are retained.

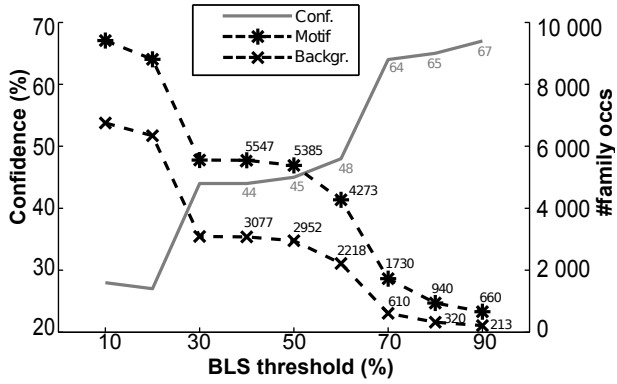


Figure 6.13: Confidence chart for the reference motif TGANNGANNGAN as a function of BLS threshold, along with the number of gene families in which it is conserved $F(T)$. Also, the number of conserved instances is shown for the control motifs (median value) $F_{bg}(T)$. This motif is a variant of the binding site in Zea Mays which will be discussed in the result section.

Interpretation of conservation metrics Note that the branch length score thresholds T_i on the one hand and conserved family count threshold F_{thres} and confidence score threshold C_{thres} on the other hand are independent. The former provides information about the degree of conservation within a single gene family whereas the latter are indicative of the degree of genome-wide conservation. Certain motifs only

show up as being genome-wide conserved for high BLS thresholds. This is typically the case for short and/or highly degenerate motifs, where also permutations of that motif are conserved with a moderate BLS in a rather large number of families, resulting in a low confidence score C. Conversely, a lower BLS threshold allows for the detection of longer motifs with genome-wide conservation in only a subset of the species. Using only a single BLS threshold would therefore limit the sensitivity of the method.

The result of the BLSSpeller algorithm is thus a *motif-confidence chart* for every motif satisfying both thresholds, as depicted in Figure 6.13.

To conclude, it might be interesting to further explore a more general set of pattern mining algorithms as introduced in the background box below:

BACKGROUND: Frequent Itemset Mining

While it might not be immediately obvious, the approach to motif discovery shares many ideas with the field of (sequential) pattern mining and Association Rule (AR) mining. A motif would there be considered an ordered sequence of events. An interesting observation is that for example the use of a *support* is a very common concept in AR mining, while the F_{thres} proved less common in motif discovery. This field of research also offers a plethora of metrics which can be used for motif discovery, such the *confidence* or the *lift* of a pattern which are metrics for evaluating the interestingness of a motif w.r.t. its prefixes or substrings[a].

[a] https://en.wikipedia.org/wiki/Association_rule_learning

Parallelization with MapReduce

Two parallel versions of the BLSSpeller algorithm have been designed. The first version relied on MPI, the second version was embedded in the MapReduce framework. The second version of the algorithm circumvented the issues which prevented the first version from scaling out to larger pattern search spaces.

MPISpeller: Lessons learnt

The initial implementation of the BLSSpeller algorithm, dubbed MPISpeller, was written in C++ and parallelized using MPI. Inter-node communication in the MPI-Speller algorithm is limited to the *sorting phase* between the intrafamily and the interfamily step. Both of these steps can be performed without any communication.

This sorting phase is responsible for making that all motifs which belong to the same permutation group end up in the same compute node. This is necessary to be able to calculate F_{bg}, which requires access to $F(T_i)$ for random permutations of a motif. For this communication we relie on the `MPI_Alltoallv` routine.

In Figure 6.14 the full parallel version of BLSSpeller is depicted. In the right-top panel we see how $F(T)$ is exported per motif: as an array with zeros and ones, with every position i in the array corresponding to $F(T_i)$ for a motif in a single gene family.

Runtime evaluation MPISpeller was described in a research paper [12] presented at the international conference on Parallel Processing and Applied Mathematics (PPAM 2014).

In this work we evaluated the algorithm on the Monocot dataset (500 bp promoter sequences instead of 2 kbp) using the ACGT+N alphabet with $e_{max} = 3$. The full MPI cluster consisted of 96 parallel processes, meaning every node has to process 185 families during the intrafamily phase. This assignment is performed by sorting the gene families according to S_{tot}, the total sequence length, and assigning each consecutive family to a node which has received the lowest amount of work so far (work is defined as the sum of the S_{tot}).

85% (48 minutes) of the total runtime is spent in the intrafamily phase of the algorithm. The load balancing performs well. All individual processes finish within a 5 minute window: the first process finishes in 2603 seconds, while the last one finishes in 2908 seconds. Within this step 94% of the time is used for running the speller algorithm and 6% for the construction of GSTs.

The next step is the sorting step. All data is partitioned such that the motifs belonging to the same permutation group end up in the same target process.

In order to obtain a uniform workload distribution during the inter-family phase, we assign a weight W_g to each permutation group g that corresponds to the maximum number of words represented by this permutation group:

$$W_g = \frac{(n_A + n_C + n_G + n_T + n_N)!}{n_A!\, n_C!\, n_G!\, n_T!\, n_N!},$$

where n_X denotes the number of characters X in a word of the permutation group. This weight is used to attribute roughly the same number of words to each process.

The total time for the sorting phase is 14% (8 minutes). Half of this time is used for the network communication while the other half is used for packing and unpacking

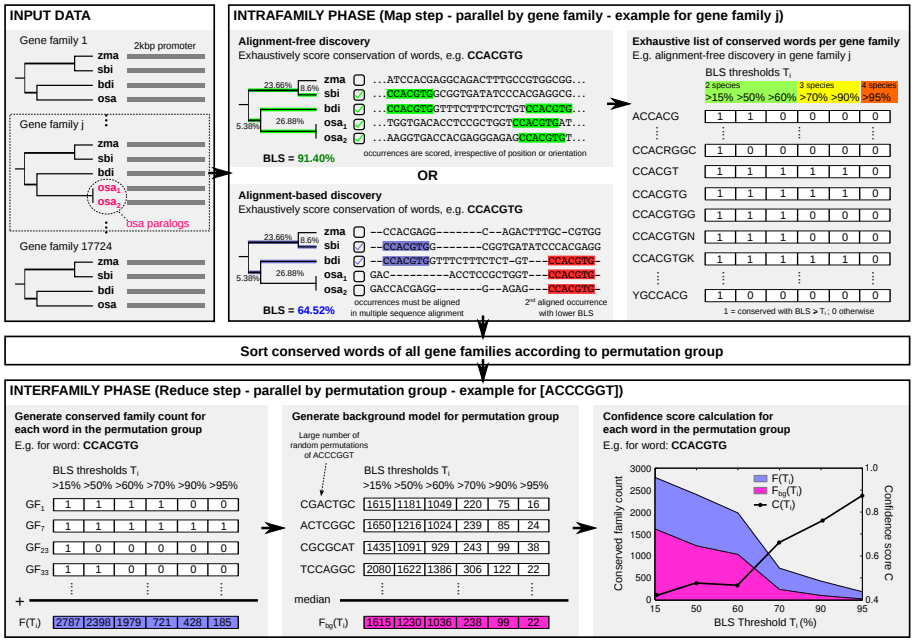


Figure 6.14: Overview of the parallel BLSSpeller algorithm.
Top: The input consists of homologous promoter sequences grouped into gene families. During the *intrafamily phase*, conserved words are exhaustively enumerated for each gene family individually. A word is considered to be conserved in a gene family if its branch length score (BLS) exceeds threshold T. Multiple BLS thresholds T_i can be used in a single run. In the *alignment-free* mode, the BLS of a word is computed irrespective of its orientation or relative position within the promoter sequences. Alternatively, in the *alignment-based* mode, words must appear aligned in the multiple sequence alignment.
Center: During the *sorting phase*, conserved words of all gene families are sorted according to permutation group, i.e., words with the same length and base content are grouped together.
Bottom: In the *interfamily phase*, permutation groups are handled individually. First, for each word, the conserved family count $F(T_i)$, i.e., the number of gene families in which the word is conserved with BLS $\geq T_i$, is established for all BLS thresholds T_i. Next, a background model $F_{\text{bg}}(T_i)$ is created by selecting the median value of the conserved family count of a large number of randomly generated instances of the permutation group, again for each threshold T_i. Finally, a confidence score $C(T_i)$ is computed for each T_i. Words for which $F(T_i) \geq F_{\text{thres}}$ and $C(T_i) \geq C_{\text{thres}}$ for any threshold T_i are considered to be genome-wide conserved motifs and are retained.

of the motif frequency vectors (representing F(T)). The final inter-family step only takes 20 seconds of the time.

As only the time spent in the sorting phase is overhead (compared to the single process execution) the algorithm is expected to exhibit a significant speedup. In Figure 6.15 the speedup is shown for different runs of the algorithm going from 1 to 256 processes, with all measurements for $P = 2^i$ $(i = 1 \ldots 8)$.

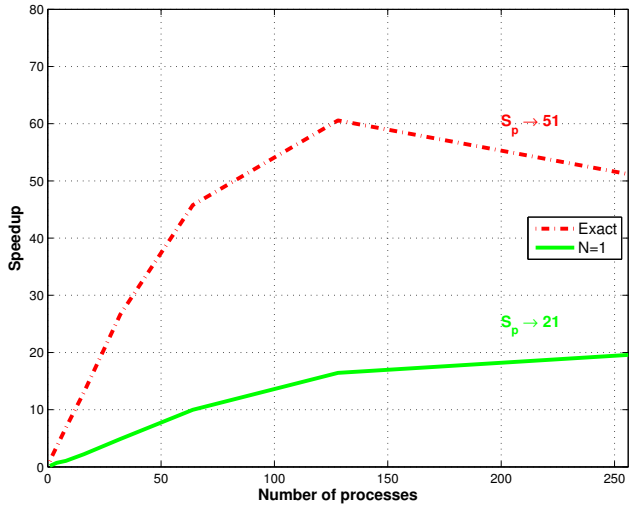


Figure 6.15: The parallel speedup of MPISpeller as a function of the number of processes. For increasing e_{max} (= the amount of 'N' wild-cards) the maximal speedup quickly goes down which leads to the conclusion that the main advantage of adding nodes to the MPISpeller parallel implementation is to have a larger total memory pool to store intermediate results.

Limits of MPISpeller The MPISpeller algorithm has a number of shortcomings, which quickly led to the conclusion it had the be abandoned in its current form: The algorithm's memory constraints prevent it from handling the more flexible ACGT+RYSWKM alphabet. This is because the amount of motifs emitted by a single family is much higher for this type of alphabet. In Figure 6.16 the number of motifs emitted per gene family is shown for these two alphabets.

A limited pattern space has the additional advantage that the F/P gene families processed by a single node in the intrafamily phase produce many common motifs which can be aggregated prior to the sorting phase. For larger pattern spaces this probability of having a common motif quickly drops to zero. This can be intuitively

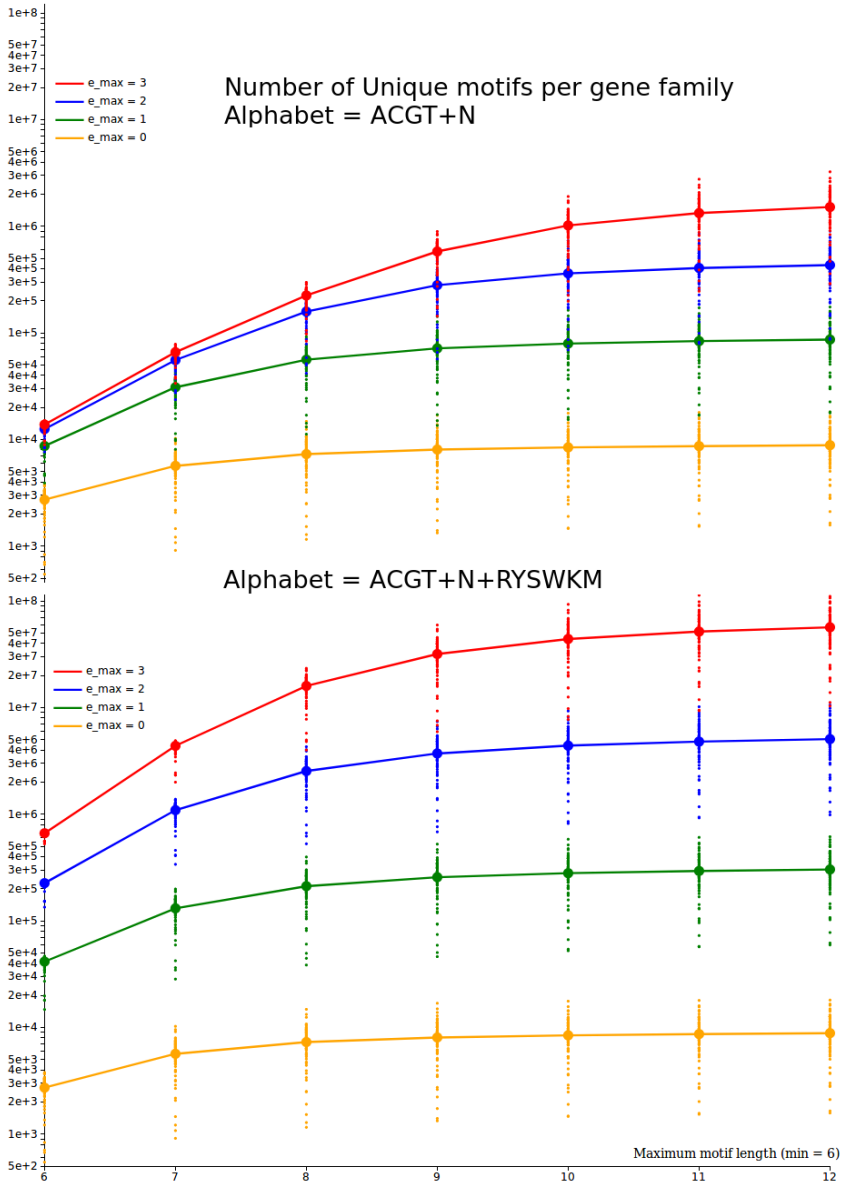


Figure 6.16: The cumulative number of unique motifs per gene family (2kbp promoters) for the ACGT+N alphabet (top) and the ACGT+RYSWKM alphabet (bottom). Note that for $k_{max} = 12$ the difference between the alphabet is already a factor 40: 1.51 million motifs (top) versus 56.6 million motifs (bottom). The curves consist of two phases, the left side corresponds to exponential growth in terms of k, the right side correspond to a saturation caused by the limited sequence length S_{tot}

seen by comparing the amount of motifs reported in a single gene family to the maximum number of motifs. If this fraction is close to 1 then the intersection of the output of two gene families will also be very large. The more flexible the alphabet the smaller this fraction becomes, for the longest motifs with $e = 3$, which are the highest in number by an order of magnitude the overlap is negligible:

The search space for k_{max} and e_{max} contains 29 billion patterns. In Figure 6.16 we show that the actual number of motifs per gene family is 57 million. The probability for a random pattern to be shared between two gene families is therefore $p_{common} \approx 0.002$.

As a result of this memory constraints and the inflexible static load balancing scheme typically only one single node had to run into problems (swapping). This leads to a significant drop in performance and typically the remaining $P-1$ nodes are idle, totally eliminating any parallel speedup. Note however that while the static load balancing is clearly a limitation of the MPI version of BLSSpeller, it is not the primary concern. If the memory constraints could have been resolved the load balancing might have become the bottleneck, but the tests with a small pattern space in the previous paragraphs indicate that the amount of idle time per node was rather small ($T_{idle} < 10\%$)

Summarized, shortcomings are:

- Limiting the algorithm to *only work in main memory* proves to lay a severe constraint on the pattern space that can be explored. It is therefore recommended to consider using secondary-storage as well.
- As load balancing is critical, the choice for *static balancing* is a clear weakness of the algorithm.
- Not mentioned so far is the increasing probability of node failures. Long-running simulations of MPISpeller on 200 processes failed multiple times because the software is not resilient against individual *node failures*.

MapReduce Implementation

A new version of the BLSSPeller algorithm was implemented using the MapReduce [17] programming model. The map phase corresponds to the intrafamily phase in which the gene families are processed in parallel by the different *mappers*. The reduce

phase corresponds to the interfamily phase in which the permutations groups are processed in parallel by the different *reducers*. In between the map and reduce step, the candidate motifs are sorted according to length and base content, in order to create the permutation groups.

The map and reduce phase of the BLSSpeller algorithm are shown in Figure 6.14. Note that the MapReduce framework typically works with shuffling and transforming sets of key-value pairs. The intermediate (map output) keys used for the BLSSpeller algorithm are therefore identifiers for the permutation groups, obtained by lexicographically sorting the characters in a motif.

Addressing the shortcomings of MPISpeller The MapReduce framework is perfectly suited for the motif discovery task as it bares a lot of similarity with the standard algorithm for building an inverted index. The MapReduce framework also solves all shortcomings of the previous MPI-Speller implementation:

1. All output of the map phase is **written to disk** lifting the memory constraint.
2. The load balancing is **dynamic**.
3. Node **failures** are monitored and dealt with, by the master node (JobTracker).
4. The MapReduce framework can deal with **stragglers** (slow executing nodes) by means of speculative execution.
5. The **shuffle phase** does not have to wait for the mappers to finish. Instead the mapper output is already copied during the map phase.
6. The MapReduce version of BLSSpeller is written in Java. This alleviates the burden of manual **memory (de-)allocation**.

With this choice of framework suddenly the ACGT+RYSWKM alphabet could be used, seriously increasing the scope of problems for which this algorithm might be useful.

The pseudo-code for the Map and Reduce function are given below:

Map function

```
def map(id, genefamily):
      GST = constructGST(genefamily.sequences)
      while GST.motif_iterator.hasnext():
            motif = GST.motif_iterator.next()
            BLS = computeBLS(motif.bitvector)
            FV = BLS.toFrequencyVector()
            key = sorted(motif)
            value = (motif, FV)
            emit(key,  value)
```

Reduce function

```
def reduce(key, mofif_list):
      H = {}
      for motif, FV in motif_list:
            H[motif] += FV
      Background = computeBG(H)
      for motif, FV in H:
            C = compute_confidence(FV, Background.FV)
            if C > 0.9:
                  emit(motif,  FV)
```

6.3 Results

The results section will start with discussing the setup and performance of the BLSSpeller algorithm. Next, we will focus on the statistical validation in a section on the False Discovery Rate. Then the focus shifts to the biological validation with a section on OCR-regions and a section on the KN1 transcription factor binding site. Finally, BLSSpeller is compared to the its closest competitors.

General results of BLSSpeller on the Monocots dataset

Dataset BLSSpeller was run on the monocots dataset described on page 121. The dataset consists of 17 724 gene families each containing four orthologous genes (one from each organism). Additionally, 10 636 paralogs are taken into account. Hence, a total of 163 064 regulatory sequences (forward and reverse strands) with a length of 2 kbp each, make up the full dataset.

As the MapReduce framework requires all data to be packed in 'records' the original dataset was preprocessed such that we have a small set of gene family records per

file. The input format for a single gene family is shown in Figure ??. The entire dataset is again not challenging in terms of volume, as it only requires storage space in the order of 100 megabytes.

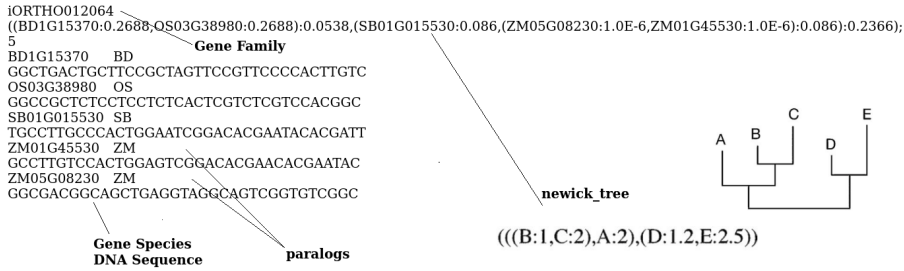


Figure 6.17: Input format for a single gene family with 5 genes, including one paralog duplication for Zea Mays. The format is a combination of the Newick format for describing trees and the FASTA format for describing genes and sequences.

Setup BLSSpeller was run on this dataset using both the alignment-free (AF) and the alignment-based (AB) discovery mode on the Amazon Web Services (Elastic MapReduce) cloud infrastructure using 20 nodes of the type m1.xlarge[3]. On every node, 7 map tasks and 2 reduce tasks were run in parallel. The computational requirements are listed in Table 6.2.

Simulation cost Based on the Amazon pricing of 2014, the financial cost for performing these simulations amounted to 1080$ and 278$ for the AF and AB cases, respectively. Note however that this de novo algorithm has to run once for every new dataset!

Data flow After the intrafamily step and using the AF discovery mode, an aggregated number of 537 billion words were found with a BLS $\geq 15\%$ (i.e., conservation in at least two species) over all 17 724 gene families. Note that these words are not necessarily unique as the same word can be conserved in multiple gene families. Using the AB discovery mode, only 82 billion words were found with a BLS $\geq 15\%$. This is because the AB discovery mode imposes the additional constraint that words should appear aligned in the multiple sequence alignment. After the interfamily step and using $F_{\text{thres}} = 1$ and $C_{\text{thres}} = 0.5$, the number of *genome-wide conserved* motifs

[3] `https://aws.amazon.com/ec2/previous-generation/`

Table 6.2: Computational requirements of BLSSpeller using both alignment-free (AF) and alignment-based (AB) discovery on the Monocot dataset.

Computational requirements	**AF**	**AB**
Number of nodes (m1.xlarge)	20	20
Intrafamily step (Map phase) (hours)	33	10
Interfamily step (Reduce phase) (hours)	11	2
Map output records ($\times 10^9$)	537	82
Map output size (TByte)	3.77	0.53
Permutation groups	48 505	48 505
Reduce output records ($\times 10^9$)	6.6	6.3
Reduce output size (TByte)	0.46	0.41

amounted to 6.62 and 6.26 billion unique motifs, for the AF and AB discovery mode respectively.

The reason why the number of motifs is high is twofold. First, very relaxed thresholds F_{thres} and C_{thres} were used. It is computationally cheap to further filter this list using more stringent (and biologically meaningful) thresholds (see next section). A second reason is the exhaustive, word-based nature of BLSSpeller. If a word is found to be genome-wide conserved, a large number of redundant, highly similar (e.g. slightly more degenerate) variants of that word may also appear in the final output of the method.

The inflation of the amount of data during the map phase (150 MB $\mapsto$ 3.8 TB) algorithm marks this as a *rather uncommon Big Data problem*. Nonetheless, the MapReduce framework is very suitable to handle this problem, as it is able to spill the map output to disk.

Runtimes The runtime for an individual gene family as a function of k_{max} and e_{max} is shown in Figure 6.18. The similarity with Figure 6.16 is striking and gives strong evidence that the runtime of the intrafamily step is strongly correlated with the amount of motifs reported, i.e. the *map output records*. Interactive versions of

both charts can be found on the BLSSpeller project website[4,5], also for other types of motif alphabets.

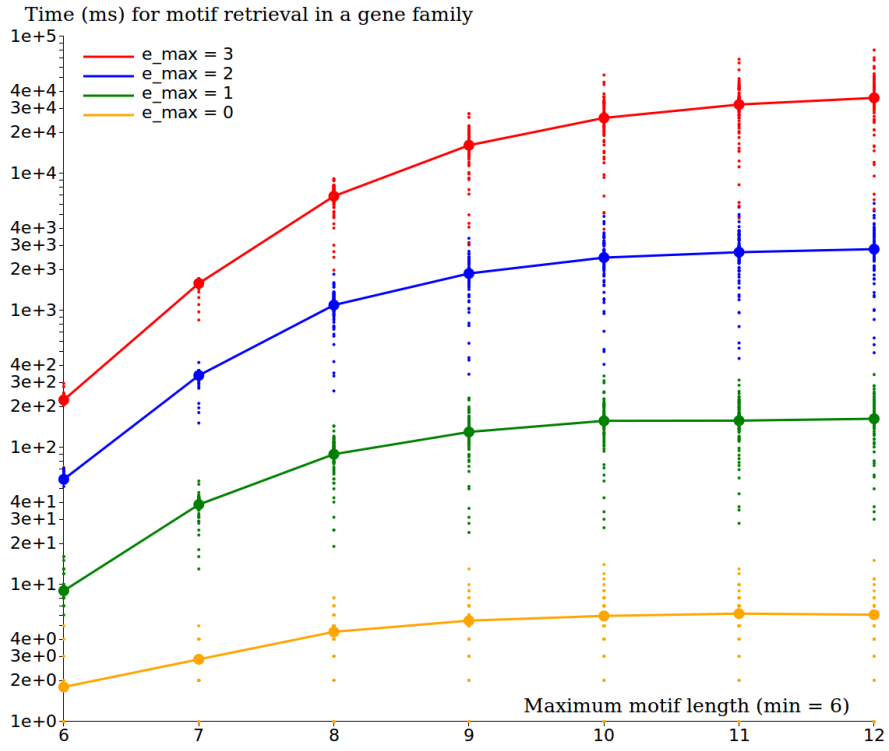


Figure 6.18: Runtime (ms) to generate all words in the restricted IUPAC alphabet (y-axis, log-scale) as a function of $k_{\max}$ (x-axis) and $e_{\max}$ (different curves) for 100 gene families. The solid lines indicate the average runtime over 100 gene families. Runs were performed on a single core of an Intel Core i7-4610M CPU @ 3.00 GHz.

Are larger datasets feasible? The simulation runtime and cost shown in Table 6.2 could be wrongfully interpreted, as if larger datasets are not feasible with the BLSSpeller algorithm. There is however an important nuance which negates this assumption: For larger phylogenetic trees the amount of motifs per gene family will roughly scale linearly as $\mathcal{O}(S_{tot})$, but having more specifies gives the end-user more freedom as to set the BLS-thresholds T_i. This was studied prior to the parallelization of the algorithm when studying the impact of the quorum constraint on the single family runtime. In Figure 6.19 we see that for a hypothetical gene family of 100 genes

[4] http://bioinformatics.intec.ugent.be/blsspeller/NumberOfMotifs.html
[5] http://bioinformatics.intec.ugent.be/blsspeller/MotifDiscoveryTime.html

increasing the quorum constraint to 10-20% speeds up the algorithm by multiple orders of magnitude. This effect is stronger for more flexible motifs with a larger e_{max}.

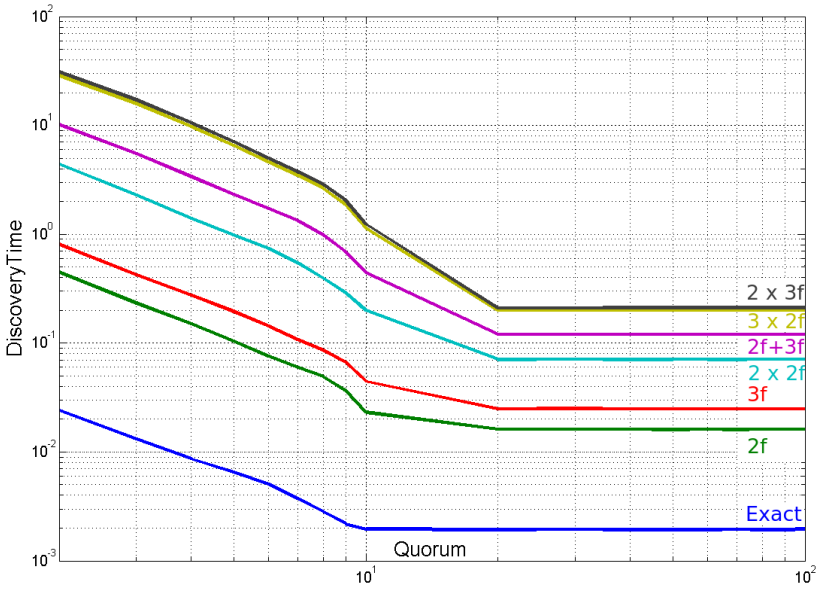


Figure 6.19: Runtime (ms) to generate all words in a set of 100 random sequences of length 250 bp. Each curve corresponds to a particular type of IUPAC patterns. (2f: the number of 2-fold degenerate characters, 3f: 3-folds deg. chars). (y-axis, log-scale) as a function of the quorum constraint.

Estimation of the False Discovery Rate (FDR)

The output of BLSSpeller consists of a list of motifs, along with the conserved family count $F(T_i)$ and conservation score $C(T_i)$ for the six different BLS thresholds T_i. This list was filtered using more stringent thresholds for F_{thres} (i.e., 1, 10 and 20) and C_{thres} (i.e., 0.5, 0.7 and 0.9). Additionally, the list can be filtered by considering only a (stricter) subset of the BLS thresholds T_i (i.e., all six thresholds $T_1, \ldots, T_6$, three thresholds $T_4, \ldots, T_6$ corresponding to conservation in at least three species, a single threshold T_6 corresponding to conservation in all four species). The number of genome-wide conserved motifs for all 27 parameter combinations is shown in Figure 6.20 for both AF and AB discovery. Clearly, each of the parameters has a strong influence on the final number of motifs in both the AF and AB discovery.

	Alignment-free discovery			Alignment-based discovery		
	BLS thresholds T_i used			BLS threshold T_i used		
C_{thres} F_{thres}	$T_1 \mapsto T_6$	$T_4 \mapsto T_6$	T_6	$T_1 \mapsto T_6$	$T_4 \mapsto T_6$	T_6
≥ 1	6.62E9 (4.09E9)	2.56E9 (4.32E8)	7.92E8 (4.57E7)	6.26E9 (3.77E8)	1.95E9 (3.47E6)	6.61E8 (1.04E5)
≥ 0.5 ≥ 10	1.08E9 (9.24E7)	1.39E8 (5.68E6)	2.74E7 (6.21E5)	4.34E8 (2.19E6)	3.68E7 (1.73E4)	7.23E6 (34)
≥ 20	5.34E8 (1.05E7)	7.55E7 (4.62E5)	1.57E7 (3.69E4)	1.47E8 (1.38E5)	1.33E7 (1.40E3)	2.54E6 (2)
≥ 1	4.98E9 (2.95E9)	2.36E9 (3.53E8)	7.31E8 (3.42E7)	5.07E9 (3.32E8)	1.86E9 (2.95E6)	6.22E8 (9.10E4)
≥ 0.7 ≥ 10	5.01E8 (1.55E7)	7.48E7 (6.50E5)	1.40E7 (3.77E4)	1.89E8 (2.73E5)	1.99E7 (1.15E3)	3.66E6 (15)
≥ 20	2.23E8 (1.15E6)	3.64E7 (6.61E3)	7.63E6 (63)	5.16E7 (3.20E3)	6.17E6 (3)	1.12E6 (0)
≥ 1	4.55E9 (2.76E9)	2.30E9 (3.45E8)	7.04E8 (3.30E7)	4.82E9 (3.26E8)	1.83E9 (2.90E6)	6.09E8 (8.99E4)
≥ 0.9 ≥ 10	9.50E7 (2.64E6)	2.16E7 (4.16E4)	4.16E6 (141)	3.79E7 (3.59E4)	6.81E6 (10)	1.34E6 (0)
≥ 20	3.85E7 (1.53E5)	8.71E6 (249)	1.77E6 (0)	8.73E6 (67)	1.89E6 (0)	3.70E5 (0)

FDR ranges
[25%, 100%]
[10%, 25%]
[5%, 10%]
[1%, 5%]
[0%, 1%]

Figure 6.20: Number of genome-wide conserved motifs for both alignment-based and alignment-free discovery for different values of C_{thres} and F_{thres} and different subsets of the six BLS thresholds T_i ($T_1 = 15\%, T_2 = 50\%, T_3 = 60\%, T_4 = 70\%, T_5 = 90\%$ and $T_6 = 95\%$). Top number: real Monocot dataset; bottom number between brackets: random dataset (zeroth-order Markov model). The colors represent the false discovery rate (see legend).

Random Datasets using Markov models In order to assess the specificity of the method for the different parameter combinations, we estimate the False Discovery Rate (FDR) in an *empirical* fashion by running BLSSpeller on a *random dataset* generated using a zeroth-order Markov model (preservation of mononucleotide frequencies) as provided by RSAT [53]. More information on the use of Markov models for random sequence generation can be found in the background box on page 145.

A number of observations can be made:

1. For comparable parameter settings, *AB discovery has a lower FDR compared to AF discovery*. The multiple sequence alignment method increases the specificity for AB discovery as relatively few words will be aligned in random data purely by chance.

2. *Low values of F_{thres} result in a poor FDR*. The reason for this is that in such case, the output consists of a large number of words that are conserved in only a single gene family. If these words are long and/or have low degeneracy, most random permutations of that word will not be conserved in any gene family, resulting in a confidence score $C(T_i) = 1$. We therefore recommend to impose a certain threshold F_{thres} on the conserved family count. As functional transcription factors typically target multiple genes, this appears to be a biologically reasonable approach.

3. A reasonable *threshold on the confidence score* should be applied. Applying this threshold, filters words for which their random permutations are conserved in a comparable number of gene families. This comprises low-complexity motifs and/or highly degenerate motifs.

4. A more stringent definition of conservation results in an *improved FDR*. This can be obtained by imposing *higher BLS* thresholds T_i.

Even though there is a clear correlation between each of the parameters and the FDR, the exact FDR is hard to predict up front and likely also depends on the dataset that is used. We therefore recommend, to run BLSSpeller with relaxed parameter settings, on both real and random data, and to filter this output using more stringent parameters until a reasonable FDR is obtained.

AF motif discovery has a higher sensitivity For reasonably stringent parameter settings where the FDR $< 1\%$, the *AF discovery mode reports 3.1 to 6.8 times more motifs* compared to the AB discovery. At first glance, this may seem to be a trivial consequence of the relaxed definition of *conservation* in the AF methodology. Indeed, a word that is found to be conserved in a gene family with BLS $\geq T$ using the AB discovery will also be conserved in the AF method. Therefore, $F^{\text{AF}}(T) \geq F^{\text{AB}}(T)$ for each word. However, in order to establish the confidence score $C(T)$, the conserved

family count $F(T)$ is compared to the corresponding median value $F_{\text{bg}}(T)$ of the background distribution. As $F_{\text{bg}}^{\text{AF}}(T)$ is also computed using the relaxed, alignment-free definition of conservation, it holds that $F_{\text{bg}}^{\text{AF}}(T) \geq F_{\text{bg}}^{\text{AB}}(T)$. Therefore, there is no reason to assume a priori, that the AF mode will pick up more motifs than its AB counterpart. This can indeed be observed in Figure 6.20 for a few parameter combinations, e.g., $F_{\text{thres}} = 1$, $C_{\text{thres}} = 0.7$ and BLS thresholds $T_1 \ldots T_6$. The reason that we do find more genome-wide conserved motifs for most parameter combinations (including those with good FDR) is because we found a significant number of known motif instances to be misaligned in this relatively highly diverged Monocot dataset.

Markov Models for Random Sequence Generation

A biological sequence can be modeled as a sequence of events. The main property of fully observable Markov model or *Markov Chain* is that the probability of the event at position t+1 depends only on the state of the chain at fixed number of previous states, the amount of states is called the order σ of the model. Markov models are translation invariant.

Let's use a small example to illustrate how a Markov chain can be used to generate random DNA sequences. Suppose we have a sequence $s = \text{ACGTG}$. Now we want to calculate the base probabilities for the next position. The probability of having an 'A' next with

$$P(s_{t+1} = A | s_t = G, s_{t-1} = T, \ldots s_0 = A) = P(s_{t+1} = A | s_t = G, \ldots s_{t-\sigma+1})$$

For a Markov model with $\sigma = 1$, the probability thus only depends on the previous character:

$$P(s_{t+1} = A | \ldots) = P(s_{t+1} = A | s_t = G) = \frac{P(AG)}{P(G)}$$

which is just the Bayes rule for conditional probabilities.

This means that a random sequence can now be generated if all possible k-mer probabilities are known, in this case: $k = 2$ and $k = 1$ are needed. For these probabilities we rely on the Maximum Likelihood Estimation.

Higher order Markov models The FDR analysis was also in run in more detail by considering higher order Markov models: first and second-order models. Whereas as zeroth-order Markov model preserves only the relative mononucleotide (A, C, G, T) occurrences, a first-order Markov model preserves also relative dinucleotide composition. Similarly, a third-order Markov models preserves both mono-, di- as well as trinucleotide composition. Tables similar to Figure 6.20 can be found in the supplementary material of the BLSSpeller paper [15]. The use of higher-order Markov models results in an increased FDR, especially for relaxed settings of BLS, C and F thresholds. However, even for the second-order Markov model, the FDR for the datasets we used in the enrichment analysis is respectively 1.11% and 2.05%, which is still very low. The same remark holds for the motifs in the KN1 analysis in section 6.3.

FDR for different motif lengths and degeneracy Additional FDR analysis can be performed as function of motif length k and degeneracy s. Here, s is defined as the total number of exact words that are implied by the degenerate word, i.e., $s = 2^{d_2} \cdot 4^{d_4}$, where d_2, and d_4 denote the number of two-fold and four-fold degenerate characters in a word, respectively.

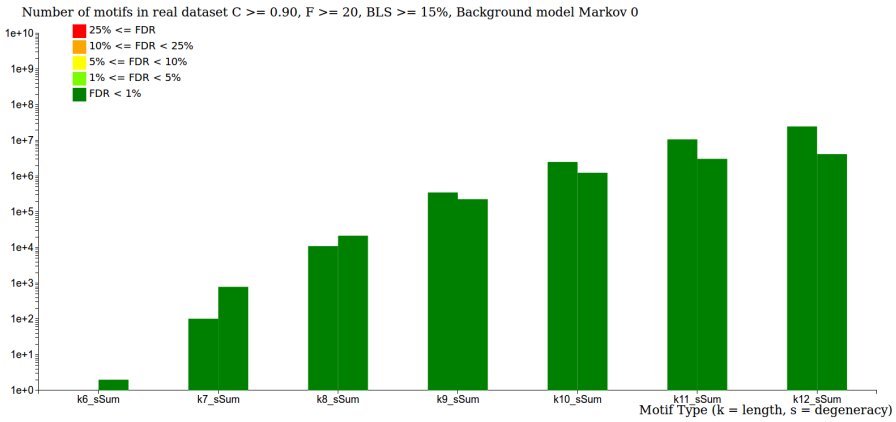


Figure 6.21: Number of motifs (y-axis, log-scale) as a function of motif length k (x-axis) for both alignment-free (left bar) and alignment-based (right bar) discovery on the Monocot dataset with $C \geq 0.9$; $F \geq 20$; BLS $\geq 15\%$. The colors represents the False Discovery Rate (FDR).

Figure 6.21 shows the number of motifs and FDR for $C_{\text{thres}} = 0.7$, $F_{\text{thres}} = 20$ and BLS $\geq 15\%$ as a function of motif length k and degeneracy s. This illustrates that the

FDR is under control for all lengths and degeneracies. An interactive version of this graph can be explored online[6]. This interactive visualization supports filtering on all thresholds, the order of the background model, the algorithm type, the motif length and degeneracy. Furthermore, aggregations on k and s can be calculated. Again, this shows the power of interactive visualizations: all unique parameter combinations correspond to over 10,000 visualizations.

Biological Validation: Motif instance predictions correlate with experimental cis-regulatory datasets

The genome-wide conserved motifs discovered by BLSSpeller are highly redundant. High-scoring motifs (AF discovery; BLS $\geq 15\%$, $C \geq 0.9$, $F \geq 20$; 38 462 976 motifs in total) were mapped back to the promoter sequences and were found to cluster around specific genomic regions. As an example we show the distribution of mapped back motifs in Figure 6.22 and Figure 6.23.

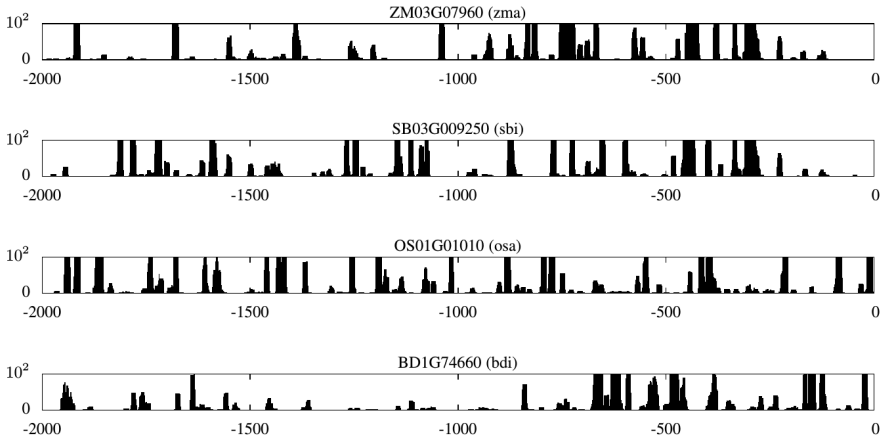


Figure 6.22: Conserved regions in the promoters of the genes in gene family iORTHO00001 corresponding to motif instances with BLS $\geq 15\%$, $F \geq 20$ and $C \geq 0.9$, i.e., high-scoring motifs that are conserved in at least two species. The height of the bars corresponds to the number of distinct motif variants that map to that location. Note that the y-axis has been truncated at 100: certain loci in this gene family are covered with up to 18 418 distinct motif variants.

Motif enrichment in OCR Certain loci are covered by thousands of highly similar motif variants. Nevertheless, the high-scoring motifs delineate distinct con-

[6] http://bioinformatics.intec.ugent.be/blsspeller/AFABHistograms.html

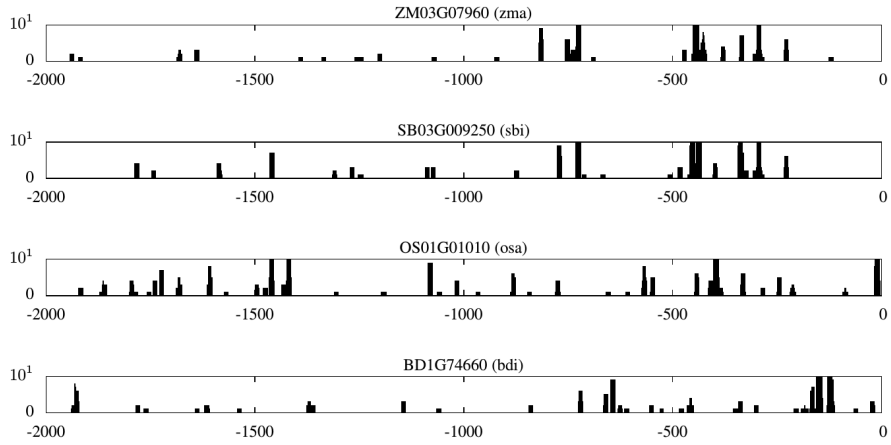


Figure 6.23: Conserved regions in the promoters of the genes in gene family iORTHO00001 corresponding to motifs instances with BLS $\geq 95\%$, $F(95\%) \geq 50$ and $C(95\%) \geq 0.9$, i.e., motifs conserved in all four species. The height of the bars corresponds to the number of distinct motif variants that map to that location. Note that the y-axis has been truncated at 10: certain loci in this gene family are covered with up to 568 distinct motif variants.

served genomic intervals on the promoter sequences. For these conserved regions, we investigated the accessibility for transcription factor binding in the promoter sequences of rice genes. *DNase I hypersensitive sites* are associated with regions of *open chromatin* where the DNA is accessible and as such provide a global perspective on possible protein-binding to the genome. Such regions were recently characterized by Zhang [64]. We performed overlap analysis between conserved genomic regions (as determined by BLSSpeller) and open chromatin regions.

The expected amount of conserved motifs in DH sites or overlapping with predicted transcription factor binding sites was determined by shuffling the conserved motif dataset 1000 times using shuffleBed across the 2kb upstream regions. The overlap was determined for each shuffled file and the median number of conserved motifs over 1000 shuffled files was used as a measure for the expected presence of conserved motifs in DH sites or overlapping with predicted transcription factor binding sites. This estimation was used to calculate the *fold enrichment*, defined as the ratio between observed overlap and expected overlap by chance.

We found a significant enrichment (3.005 fold) of conserved regions for open chromatin regions (p-value < 0.001) (see Table 6.3). For a stricter subset of motifs (AF

Table 6.3: Overlap between conserved genomic regions as identified by BLSSpeller and experimentally profiled open chromatin regions in rice and transcription factor binding sites inferred through protein-binding microarrays in rice and maize. Regions are required to fully overlap in order to be scored.

Overlap with experimentally profiled open chromatin regions (OCR) in *Oryza sativa*					
BLSSpeller thresholds	# conserved regions	# OCR regions	# conserved regions within OCR regions	# rand. conserved regions within OCR regions	enrichment fold
BLS $\geq$ 15%, $C \geq 0.9$, $F \geq 20$	754 205	77 247	121 026	40 277	3.005
BLS $\geq$ 95%, $C \geq 0.9$, $F \geq 20$	464 229	77 247	98 681	25 996	3.796
Overlap with experimentally profiled TF binding sites (TBS) in *Oryza sativa*					
BLSSpeller thresholds	# conserved regions	# TBS regions	# TBS regions within conserved regions	# TBS regions within rand. conserved regions	enrichment fold
BLS $\geq$ 15%, $C \geq 0.9$, $F \geq 20$	754 205	442 506	159 542	42 522	3.752
BLS $\geq$ 95%, $C \geq 0.9$, $F \geq 20$	464 229	442 506	37 093	5 689	6.520
Overlap with experimentally profiled TF binding sites (TBS) in *Zea mays*					
BLSSpeller thresholds	# conserved regions	# TBS regions	# TBS regions within conserved regions	# TBS regions within rand. conserved regions	enrichment fold
BLS $\geq$ 15%, $C \geq 0.9$, $F \geq 20$	828 400	482 317	156 929	66 564	2.358
BLS $\geq$ 95%, $C \geq 0.9$, $F \geq 20$	454 221	482 317	35 710	10 755	3.320

discovery; BLS $\geq 95\%, C \geq 0.9, F \geq 20$; 1 769 963 motifs in total), the fold enrichment increased to 3.796.

Enrichment in experimentally validated TF binding sites Additionally, we investigated the enrichment of TF binding sites determined in vitro [61] towards conserved genomic regions in rice and maize. Transcription factor DNA binding specificities are the primary mechanism by which transcription factors recognize genomic features and regulate genes. Recently, a dataset containing a large number of these binding specificities was generated using protein-binding microarrays (PBM) [61]. From this database, PWMs were downloaded for 481 TFs in rice and for 615 TFs in maize. These were mapped onto the respective rice and maize promoters and overlap analysis was performed. In rice, of the 754 205 constrained genomic regions (BLS $\geq 15\%$), 159 542 contain a PBM-based TF binding site, leading to 3.752 fold enrichment (p-value < 0.001). Again, for the stricter subset of conserved motifs (BLS $\geq 95\%$), fold enrichment increased to 6.520. Maize showed a fold enrichment of 2.358 and 3.320 (p-value < 0.001) respectively. Overall, these analyses revealed that a large part of the conserved non-coding sequences can be accessed by DNA binding proteins and as such can act as functional transcription factor binding sites, and that these conserved non-coding sequences show enrichment for the binding sites of a large number of TFs inferred using PBMs.

Biological Validation: Conservation of KN1 binding site in Zea Mays

ChIP-seq experiments KNOTTED1 (KN1) transcription factors are involved in the establishment and maintenance of plant meristems and are thought to be conserved among the family of grasses [5]. [6] profiled KN1 binding sites in *Zea mays* using ChIP-seq experiments. The overlapping loci in two samples of immature ears were retained and assigned to the nearest gene within a range of 10 kbp. The ChIP-Seq peaks were found to be mainly situated in the 5' en 3' regions extending from the gene but also occur in introns and exons. Thus, a set of 5 118 candidate KN1-regulated maize genes were identified. For approximately 7% of these genes, a binding site reminiscent of the intronic KN1 binding site in ga2ox1, was identified. For these so-called ga2ox1-like KN1 binding sites, a Position Weight Matrix (PWM) was derived by [6]. Translated to the IUPAC alphabet, this PWM corresponds to TGAYNGAYDGAY. The PWM and its corresponding logo are shown in Figure 6.24

	1	2	3	4	5	6	7	8	9	10	11	12
A			1.00		0.10		1.00		0.25		1.00	
C				0.82	0.08			0.38				0.82
G		1.00			0.32	1.00			0.25	1.00		
T	1.00			0.18	0.50			0.62	0.50			0.18

Figure 6.24: Experimental PWM for KN1 as derived by Bolduc. Sequence logos show the information content I per position. The height of the individual bases is obtained by multiplying with the base probabilities: $f_b \times I$

Overlap analysis with BLSSpeller AF motifs We investigate whether BLSSpeller is able to discover the ga2ox1-like KN1 motifs and binding sites through a comparative study of the four monocot species. From the BLSSpeller output, all genome-wide conserved motifs of length 12 that match the ga2ox1-like KN1 PWM identified by [6] were retained. Using alignment-free discovery, and using $F_{\text{thres}} = 20$ and $C_{\text{thres}} = 0.7$ (FDR $\leq 1\%$, see Figure 6.20), 51 genome-wide conserved motif variants are identified. In total, these motifs are conserved in 165 gene families with a BLS $\geq 15\%$ (i.e., conservation in at least two species). From the 51 identified motif variants, only 19 are required to explain the conservation in all 165 gene families. These essential motifs are listed in Table 6.4 along with their respective metrics. In turn, these gene families contain 213 maize genes in total, 51 of which were also identified in Bolduc [6]. These results were compared to those obtained by Fastcompare [20], a method that also performs motif discovery in an alignment-free

and exhaustive manner. However, Fastcompare is limited to the exact ACGT alphabet and pairwise species comparisons. Because of these limitations, Fastcompare could identify only 36 maize gene targets, 10 of which were also identified by Bolduc [6]. The comparison with similar methods is the subject of the next section.

Overlap analysis with BLSSpeller AB motifs Similarly, using the BLSSpeller AB discovery mode, conservation with a BLS $\geq 15\%$ is observed in only 37 gene families, even with very relaxed thresholds ($F_{\text{thres}} = 1$ and $C_{\text{thres}} = 0.7$) (FDR $\leq 10\%$). The 9 essential motif variants required to explain this conservation are listed in Table 6.4. The 37 gene families contain 41 maize genes, 10 of which are also reported in [6]. Inspection of the promoter sequence alignments of the gene families reveals that the ga2ox1-like KN1 variants are often not aligned, either because the motif instances in the different species are located at entirely different positions in the promoter sequences or because they appear on different strands. Therefore, alignment-based motif discovery approaches such as BLSSpeller in AB mode or the 'mini motifs' approach as used by [49] suffer from reduced sensitivity on diverged datasets.

Comparison to other similar Motif Discovery Approaches

BLSSpeller was compared to three alternative methods:

1. **Fastcompare** [19] is similar to BLSSpeller in the sense that it is alignment-free and exhaustive. It is limited compared to BLSSpeller in the sense that it is restricted to pairwise comparisons between species and limited to the ACGT alphabet.

2. **MDOS** [62] is very similar to Fastcompare: it is alignment-free, exhaustive over the exact (i.e., ACGT) alphabet and relies on pairwise species comparisons. There are three main differences. First, whereas Fastcompare, like BLSSpeller, does not take the strand into consideration when scoring conservation, MDOS does. Second, MDOS does not rely on the hypergeometric distribution to compute the p-value but uses a more sophisticated technique to score significance of conservation. Third, after all motif instances in exact alphabet are scored in an exhaustive manner, MDOS tries to insert degenerate characters in top scoring motifs to check whether this could improve the conservation significance.

Table 6.4: List of genome-wide conserved ga2ox1-like KN1 motif variants identified by BLSSpeller using both AF and AB discovery. $F(15\%)$ denotes the number of gene families in which the motif is conserved with BLS $\geq 15\%$ while $C(15\%)$ denotes the corresponding confidence score. $\mathcal{M}_{\text{BLS}}$ denotes the number of maize genes contained in the gene families while $\mathcal{M}_{\text{inters}}$ denotes the intersection $\mathcal{M}_{\text{BLS}} \cap \mathcal{M}_{\text{ChIP}}$ with experimentally profiled maize genes.

Alignment-free discovery					Alignment-based discovery				
KN1 motif variant	$F(15\%)$	$C(15\%)$	$\mathcal{M}_{\text{BLS}}$	$\mathcal{M}_{\text{inters}}$	KN1 motif variant	$F(15\%)$	$C(15\%)$	$\mathcal{M}_{\text{BLS}}$	$\mathcal{M}_{\text{inters}}$
TGATNGATKGAY	59	0.93	75	24	TGATNGAYGGAY	11	0.91	10	3
TGATNGAYKGAT	59	0.93	74	20	TGATNGATKGAY	11	0.82	11	3
TGAYNGATKGAT	54	0.93	68	21	TGAYNGACKGAC	10	0.90	11	3
TGATNGAYWGAT	40	0.88	50	11	TGAYGGAYGGAY	9	1.00	9	3
TGAYNGAYTGAT	36	0.89	48	11	TGATNGAYRGAT	9	0.89	10	3
TGAYTGAYTGAY	33	0.97	42	9	TGAYNGAYTGAC	8	0.88	9	2
TGATNGAYTGAY	32	0.88	40	7	TGACNGAYTGAY	8	0.88	10	3
TGAYNGATWGAT	31	0.84	42	12	TGACNGACWGAY	7	0.86	7	2
TGATNGATWGAY	30	0.83	36	9	TGACAGAYRGAY	3	1.00	4	0
TGATNGATRGAY	29	0.86	39	9					
TGAYNGATRGAT	27	0.85	37	9					
TGATNGAYRGAT	26	0.85	35	8					
TGAYNGATTGAY	25	0.84	34	7					
TGAYNGATGGAY	24	0.88	35	9					
TGATNGAYGGAY	24	0.88	31	8					
TGAYTGAYWGAT	22	0.91	27	6					
TGAYNGACTGAY	22	0.91	28	9					
TGAYNGAYTGAC	21	0.90	27	8					
TGAYNGACKGAC	20	0.90	25	10					
Union (all variants)	165	–	213	51	Union (all variants)	37	–	41	10

3. '**Mini motifs**' approach used by Stark [49] relies on multiple sequence alignments and uses the BLS metric to first find highly conserved "mini motif cores" of the form ABC-gap-XYZ, where A, B, C, X, Y, Z are characters from the ACGT alphabet and "gap" takes a variable length between 0 and 10 nucleotides. Highly conserved mini motif instances are then extended into "full motifs" according to preferential conservation in the gap region and within 5 nucleotides on both sides of the motif cores. Additionally, IUPAC characters can be inserted if that improves the hypergeometric p-value.

Issues with competing methods All three methods had some issues but we did manage to make a comparison with Fastcompare.

1. **Fastcompare** was not designed to handle motifs with a length up to 12 char-

acters: the program required over 96 GB of RAM and then exited (out of memory). Inspection of the source code revealed that this was due to a programming error. This error (a **memory leak**) was fixed and a patch was corresponded to the authors of Fastcompare.

2. We ran **MDOS** on all pairwise species combinations with zma and again examined the output for ga2ox1-like KN1 motifs. MDOS *did not find a single motif instance*. We believe this is due to the fact that MDOS only outputs results for which the motif is conserved in at least 5 gene families (this is an empirical observation based on the fact that no matter what we try, the output contains only motif variants conserved in 5 or more gene families). Additional investigation revealed that there is **not a single ga2ox1-like KN1 motif variant** (in exact ACGT alphabet) that is conserved in 5 or more gene families on the same strand. Note that TGACTGACTGAC is conserved in exactly 5 gene families, but in two of them, the occurrences are on a different strand and hence this conservation is not scored by MDOS. As (a) the assumed threshold on the number of gene families is not documented, (b) the source code of MDOS is not available (only a compiled Java .jar file) and (c) the authors of MDOS did not reply to our inquiry about this threshold, we were unable to perform a fair comparison to MDOS (other than reporting "MDOS does not find anything").

3. Because the **software** of Stark [49] **is not publicly available**, a direct comparison cannot be performed. However, we believe that there are **t**hat this method will not yield satisfactory results for the ga2ox1-like KN1 case, because it inherently relies on multiple sequence alignments. We examined, in the 165 gene families in which a ga2ox1-like KN1 was genome-wide conserved according to BLSSpeller (alignment-free methodology), **to what extent the motif instances are aligned** (or not aligned) in the Multiple Sequence Alignment (MSA):

    - In 47 gene families (28% of the cases), there is at least one pair of motif instances that align.
    - In 6 gene families (4% of the cases), there is at least one pair of motif instances that is partially aligned (i.e., the instances overlap but not all characters are aligned).

- In 61 gene families (37% of the cases), no pair of motif instances is aligned (or overlaps).
- In the other 51 gene families (31% of the cases), the *motif instances could not be aligned* because they occur on different strands. We therefore assume that the approach by Stark et al. will at best retrieve 32% of the gene families reported by BLSSpeller using the alignment-free discovery.

Results of Fastcompare After fixing the memory leak, we ran Fastcompare on all pairwise species combinations with Zea mays, shortly zma, (i.e., zma-sbi, zma-bdi and zma-osa) and examined the output for ga2ox1-like KN1 motifs. The results are presented in Table 6.5. Due to the fact that the algorithm is limited to ACGT alphabet, very few ga2ox1-like instances were found to be conserved. The best scoring motif variant, TGACTGACTGAC was conserved in only 5 gene families. Most variants were conserved in only a single gene family.

Fastcompare was run with $k = 12$ and produces as output a ranked list of motifs from which ga2ox1-like KN1 motifs were filtered. Motif variants that are conserved in at least one gene family are listed in Table 6.5. Only few variants were conserved in more than one gene family, again illustrating the fact that degeneracy in the motif model is essential for a sensitive detection of motifs in diverged species. Most variants were found to be conserved in zma and sbi, the two most closely related species in the dataset. In total, over all species combinations, 36 unique maize genes were identified in which a ga2ox1-like KN1 motif was conserved, 10 of which overlap with the experimentally profiled maize genes. Note that no multiple hypothesis correction was applied to the *p*-values.

None of the three methods mentioned is able to reproduce the results of BLSSpeller, which demonstrates the novelty and the validity of this algorithm.

Table 6.5: List of 25 ga2ox1-like KN1 motif variants retrieved by Fastcompare using alignment-free discovery in zma vs. sbi, zma vs. bdi and zma vs. osa. Here, F denotes the number of gene families in which the motif variant is conserved. $\mathcal{M}_{\text{FC}}$ denotes the number of (unique) maize genes contained in the gene families while M_{inters} denotes the intersection $M_{\text{FC}} \cap M_{\text{ChIP}}$ with experimentally profiled maize genes.

KN1 motif variant	species	F	p-value	rank	$\mathcal{M}_{\text{FC}}$	$\mathcal{M}_{\text{inters}}$
TGACTGACTGAC	zma-sbi	5	6.52e-08	78705	5	2
TGATGGATGGAT	zma-sbi	5	4.73e-06	182102	4	2
TGACAGACTGAC	zma-sbi	2	2.89e-05	218551	2	1
TGACCGACTGAC	zma-sbi	2	4.19e-05	233416	2	0
TGATTGATTGAT	zma-sbi	5	0.000144	284771	5	0
TGATCGACAGAT	zma-sbi	1	0.000303	365985	1	1
TGACCGACAGAC	zma-sbi	1	0.000303	376590	1	0
TGACAGACGGAC	zma-sbi	1	0.000379	409007	1	0
TGACCGATGGAC	zma-sbi	1	0.000569	468802	1	0
TGATAGACAGAT	zma-sbi	1	0.00102	564663	1	0
TGACTGATTGAT	zma-sbi	1	0.00121	597227	1	0
TGATGGACGGAC	zma-sbi	1	0.0019	680266	1	1
TGACAGATTGAC	zma-sbi	1	0.00227	707550	1	0
TGACAGATGGAT	zma-sbi	1	0.00273	735887	1	0
TGATTGATGGAC	zma-sbi	1	0.00545	822388	1	1
TGATTGACTGAT	zma-sbi	1	0.00583	828114	1	0
TGATGGATGGAC	zma-sbi	1	0.00816	856784	1	0
TGATTGACAGAT	zma-sbi	1	0.00907	865156	1	0
TGATCGATGGAT	zma-sbi	1	0.00982	870353	1	0
TGACTGACTGAT	zma-sbi	1	0.0163	897439	1	0
TGATTGATGGAT	zma-sbi	1	0.0185	902941	1	1
TGACTGACTGAT	zma-bdi	1	0.00855	63787	1	1
TGACTGACTGAC	zma-bdi	1	0.102	77976	1	1
TGATGGATGGAT	zma-osa	2	0.0234	80796	2	1
TGACTGACTGAC	zma-osa	1	0.0615	86863	1	1
Union (all variants)	–	39	–	–	36	10

6.4 Conclusion

A novel phylogenetic footprinting approach was developed for the sensitive discovery of conserved cis-regulatory elements, even in diverged sequences. Using IUPAC strings as motif model and using the MapReduce programming model to enable distributed computing, it was shown that it is feasible to compute all genome-wide conserved words in a large dataset, in an exhaustive manner.

For a given false discovery rate, it was demonstrated that an alignment-free approach detects more conserved words than an alignment-based approach. This proves that this version of the algorithm is more sensitive and that the extra words cannot be attributed to a difference in FDR.

Even though millions of genome-wide conserved motifs were identified by our method, mapping of these motifs to the promoter sequences results in constrained conserved genomic regions. It was shown that these conserved regions were significantly enriched for experimentally profiled open chromatin regions in rice and for TF binding sites inferred through protein-binding microarrays in rice and maize. The fact that many motifs map to the same loci also demonstrates that the actual number of predicted binding sites is much lower.

Finally, it was shown that the especially alignment-free approach shows an improved recovery of the ga2ox1-like KN1 binding site, compared to the alignment-based approach or competing methods.

References

[1] T. L. Bailey, M. Boden, F. A. Buske, M. Frith, C. E. Grant, L. Clementi, J. Ren, W. W. Li, and W. S. Noble. MEME Suite: tools for motif discovery and searching. *Nucleic Acids Research*, 37(suppl 2): W202–W208, July 2009.

[2] J. Benntzin and M. Freeling. Grasses as a single genetic system: genome composition, collinearity and compatibility. *Trends in Genetics*, 9(8):259–261, 1993.

[3] E. Berezikov, V. Guryev, R. H. Plasterk, and E. Cuppen. CONREAL: conserved regulatory elements anchored alignment algorithm for identification of transcription factor binding sites by phylogenetic footprinting. *Genome research*, 14(1):170–178, January 2004.

[4] M. Blanchette and M. Tompa. Discovery of Regulatory Elements by a Computational Method for Phylogenetic Footprinting. *Genome Research*, 12(5):739–748, May 2002.

[5] N. Bolduc and S. Hake. The maize transcription factor knotted1 directly regulates the gibberellin catabolism gene ga2ox1. *Plant Cell*, 216:1647–1658, 2009.

[6] N. Bolduc, A. Yilmaz, M. K. Mejia-Guerra, K. Morohashi, D. O'Connor, E. Grotewold, and S. Hake. Unraveling the KNOTTED1 regulatory network in maize meristems. *Genes and Development*, 2615: 1685–1690, 2012.

[7] R. K. Bradley, X.-Y. Y. Li, C. Trapnell, S. Davidson, L. Pachter, H. C. C. Chu, L. A. Tonkin, M. D. Biggin, and M. B. Eisen. Binding site turnover produces pervasive quantitative changes in transcription factor binding between closely related Drosophila species. *PLoS biology*, 8(3):e1000343+, March 2010.

[8] C. S. Carmack, L. McCue, L. Newberg, and C. Lawrence. PhyloScan: identification of transcription factor binding sites using cross-species evidence. *Algorithms for Molecular Biology*, 2(1):1+, January 2007.

[9] K. Chen, D. Durand, and M. Farach-Colton. Notung: a program for dating gene duplications and optimizing gene family trees. *Journal of Computational Biology*, 7(3-4):429–447, 2000.

[10] A. Cornish-Bowden. Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic acids research*, 13(9):3021–3030, May 1985.

[11] M. Das and H. K. Dai. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, 8(Suppl 7):S21+, 2007.

[12] D. De Witte, M. Van Bel, P. Audenaert, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. A parallel, distributed-memory framework for comparative motif discovery. *Parallel Processing and Applied Mathematics*, pages 268–277, 2013.

[13] D. De Witte, M. Van Bel, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. A high performance computing approach to the dicovery of conserved motifs. eng. *20e Annual Conference on Intelligent Systems for Molecular Biology, Abstracts*, pages 1–1, Berlin, Germany, 2012.

[14] D. De Witte, M. Van Bel, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. Alignment-free genome-wide comparative motif discovery in 4 monocot species. eng. *11th European Conference on Computational Biology, Abstracts*, pages 1–1, Basel, Switzerland, 2012.

[15] D. De Witte, J. Van de Velde, D. Decap, M. Van Bel, P. Audenaert, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. Blsspeller: exhaustive comparative discovery of conserved cis-regulatory elements. eng. *BIOINFORMATICS*, 31(23):3758–3766, 2015.

[16] D. De Witte, J. Van de Velde, M. Van Bel, P. Audenaert, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. Comparative motif discovery in the cloud. eng. *Benelux Bioinformatics Conference 2013, Abstracts*, Brussels, Belgium, 2013.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Operating System Design and Implementation*, pages 137–150, 2004.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1977.

[19] O. Elemento and S. Tavazoie. Fastcompare: a nonalignment approach for genome-scale discovery of DNA and mRNA regulatory elements using network-level conservation. *Methods Mol Biol*, 395: 349–366, 2007.

[20] O. Elemento and S. Tavazoie. Fast and systematic genome-wide discovery of conserved regulatory elements using a non-alignment based approach. *Genome Biology*, 6(2):R18+, 2005.

[21] A. Enright, S. Van Dongen, and C. Ouzounis. Tribemcl: an efficient algorithm for large scale detection of protein families. *Retrieved November*, 21:2005, 2004.

[22] E. Eskin and P. A. Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics (Oxford, England)*, 18 Suppl 1, 2002.

[23] L. Ettwiller, B. Paten, M. Souren, F. Loosli, J. Wittbrodt, and E. Birney. The discovery, positioning and verification of a set of transcription-associated motifs in vertebrates. *Genome biology*, 6(12), 2005.

[24] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In: *Readings in Computer Vision*, pages 564–584. Elsevier, 1987.

[25] R. Giegerich, S. Kurtz, and J. Stoye. Efficient Implementation of Lazy Suffix Trees. In: *International Workshop on Algorithm Engineering*, pages 30–42. Springer-Verlag, 1999.

[26] R. Gordân, L. Narlikar, and A. J. Hartemink. Finding regulatory DNA motifs using alignment-free evolutionary conservation information. *Nucleic Acids Research*, 38(6):e90, April 2010.

[27] D. Gusfield. Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge university press, 1997.

[28] J. van Helden, A. F. Rios, and J. Collado-Vides. Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Res*, 28(8):1808–1818, April 2000.

[29] J. D. Hughes, P. W. Estep, S. Tavazoie, and G. M. Church. Computational identification of Cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology*, 296(5):1205–1214, March 2000.

[30] M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E. S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423(6937):241–254, May 2003.

[31] L. Kumar, A. Breakspear, C. Kistler, L. J. Ma, and X. Xie. Systematic discovery of regulatory motifs in Fusarium graminearum by comparing four Fusarium genomes. *BMC Genomics*, 11(1):208+, 2010.

[32] C. E. Lawrence and A. A. Reilly. An expectation maximization (em) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function, and Bioinformatics*, 7(1):41–51, 1990.

[33] L. Li, C. J. Stoeckert, and D. S. Roos. Orthomcl: identification of ortholog groups for eukaryotic genomes. *Genome research*, 13(9):2178–2189, 2003.

[34] S. Liang, M. P. Samanta, and B. A. Biegel. cWINNOWER algorithm for finding fuzzy dna motifs. *J Bioinform Comput Biol*, 2(1):47–60, March 2004.

[35] X. Liu, D. L. Brutlag, and J. S. Liu. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 2001.

[36] L. Marsan and M. F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of computational biology*, 7(3-4):345–362, 2000.

[37] T. Marschall and S. Rahmann. Efficient exact motif discovery. *Bioinformatics (Oxford, England)*, 25(12):356–364, June 2009.

[38] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[39] G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics (Oxford, England)*, 17 Suppl 1(suppl 1):S207–S214, June 2001.

[40] D. A. Pollard, C. M. Bergman, J. Stoye, S. E. Celniker, and M. B. Eisen. Benchmarking tools for the alignment of functional noncoding DNA. *BMC bioinformatics*, 5(1):6+, January 2004.

[41] M. N. Price, P. S. Dehal, and A. P. Arkin. Fasttree 2–approximately maximum-likelihood trees for large alignments. *PloS one*, 5(3):e9490, 2010.

[42] S. Proost, M. Van Bel, L. Sterck, K. Billiau, T. Van Parys, Y. Van de Peer, and K. Vandepoele. PLAZA: A Comparative Genomics Resource to Study Gene and Genome Evolution in Plants. *The Plant Cell Online*, 21(12):3718–3731, December 2009.

[43] A. R. Reineke, E. Bornberg-Bauer, and J. Gu. Evolutionary divergence and limits of conserved non-coding sequence detection in plant genomes. *Nucleic Acids Research*, 39(14):6029–6043, August 2011.

[44] G. K. Sandve, O. Abul, V. Walseng, and F. Drabløs. Improved benchmarks for computational motif discovery. *BMC bioinformatics*, 8(1):193, 2007.

[45] R. V. Satya and A. Mukherjee. Pruner: algorithms for finding monad patterns in dna sequences. *CSB*, pages 662–665, IEEE Computer Society, April 21, 2005.

[46] D. H. Sieglaff, W. A. Dunn, X. S. Xie, K. Megy, O. Marinotti, and A. A. James. Comparative genomics allows the discovery of cis-regulatory elements in mosquitoes. *Proceedings of the National Academy of Sciences*, 106(9):3053–3058, March 2009.

[47] E. D. Siggia. Computational methods for transcriptional regulation. *Current opinion in genetics & development*, 15(2):214–221, April 2005.

[48] E. L. Sonnhammer and E. V. Koonin. Orthology, paralogy and proposed classification for paralog subtypes. *TRENDS in Genetics*, 18(12):619–620, 2002.

[49] A. Stark, M. F. Lin, P. Kheradpour, J. S. Pedersen, L. Parts, J. W. Carlson, M. A. Crosby, M. D. Rasmussen, S. Roy, A. N. Deoras, G. G. Ruby, J. Brennecke, Harvard FlyBase curators, Berkeley Drosophila Genome Project, E. Hodges, A. S. Hinrichs, A. Caspi, B. Paten, S.-W. W. Park, M. V. Han, M. L. Maeder, B. J. Polansky, B. E. Robson, S. Aerts, J. van Helden, B. Hassan, D. G. Gilbert, D. A. Eastman, M. Rice, M. Weir, M. W. Hahn, Y. Park, C. N. Dewey, L. Pachter, J. J. Kent, D. Haussler, E. C. Lai, D. P. Bartel, G. J. Hannon, T. C. Kaufman, M. B. Eisen, A. G. Clark, D. Smith, S. E. Celniker, W. M. Gelbart, and M. Kellis. Discovery of functional elements in 12 Drosophila genomes using evolutionary signatures. *Nature*, 450(7167):219–232, November 2007.

[50] G. Stormo. Dna binding sites: representation and discovery. *Bioinformatics*, 16:16–23.

[51] A. R. Subramanian, M. Kaufmann, and B. Morgenstern. DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for molecular biology : AMB*, 3(1):6+, May 2008.

[52] G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, P. Rouzé, and Y. Moreau. A Gibbs Sampling Method to Detect Overrepresented Motifs in the Upstream Regions of Coexpressed Genes. *Journal of Computational Biology*, 9(2):447–464, April 2002.

[53] M. Thomas-Chollier, O. Sand, J.-V. Turatsinze, R. Janky, M. Defrance, E. Vervisch, S. Brohée, and J. van Helden. RSAT: regulatory sequence analysis tools. *Nucleic Acids Research*, 36(suppl 2):W119–W127, July 2008.

[54] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature biotechnology*, 23(1):137, 2005.

[55] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[56] M. Van Bel, S. Proost, E. Wischnitzki, S. Movahedi, C. Scheerlinck, Y. Van de Peer, and K. Vandepoele. Dissecting Plant Genomes with the PLAZA Comparative Genomics Platform. *Plant Physiology*, 158(2):590–600, February 2012.

[57] S. Venkataram and J. C. Fay. Is transcription factor binding site turnover a sufficient explanation for cis-regulatory sequence divergence? *Genome biology and evolution*, 2:851–858, January 2010.

[58] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt. Prospects and limitations of full-text index structures in genome analysis. *Nucleic acids research*, 40(15):6993–7015, 2012.

[59] T. Wang. Identifying the conserved network of cis-regulatory sites of a eukaryotic genome. *Proceedings of the National Academy of Sciences*, 102(48):17400–17405, November 2005.

[60] W. Wei and X. Yu. Comparative Analysis of Regulatory Motif Discovery Tools for Transcription Factor Binding Sites. *Genomics, Proteomics & Bioinformatics*, 5(2):131–142, May 2007.

[61] M. T. Weirauch, A. Yang, M. Albu, A. G. Cote, A. Montenegro-Montero, P. Drewe, H. S. Najafabadi, S. A. Lambert, I. Mann, K. Cook, H. Zheng, A. Goity, H. van Bakel, J.-C. Lozano, M. Galli, M. G. Lewsey, E. Huang, T. Mukherjee, X. Chen, J. S. Reece-Hoyes, S. Govindarajan, G. Shaulsky, A. J. M. Walhout, F.-Y. Bouget, G. Ratsch, L. F. Larrondo, J. R. Ecker, and T. R. Hughes. Determination and Inference of Eukaryotic Transcription Factor Sequence Specificity. *Cell*, 158(6):1431–1443, November 2014.

[62] J. Wu, D. H. Sieglaff, J. Gervin, and X. S. Xie. Discovering regulatory motifs in the Plasmodium genome using comparative genomics. *Bioinformatics (Oxford, England)*, 24(17):1843–1849, September 2008.

[63] X. Xie, J. Lu, E. J. Kulbokas, T. R. Golub, V. Mootha, K. Lindblad-Toh, E. S. Lander, and M. Kellis. Systematic discovery of regulatory motifs in human promoters and 3[prime] UTRs by comparison of several mammals. *Nature*, 434(7031):338–345, March 2005.

[64] W. Zhang, Y. Wu, J. C. Schnable, Z. Zeng, M. Freeling, G. E. Crawford, and J. Jiang. High-resolution mapping of open chromatin in the rice genome. *Genome Research*, 22(1):151–162, January 2012.

Part III

Publication and Integration of Life Sciences Data

Chapter 7

Big Linked Data Solutions for the Life Sciences Domain

The ultimate measure of a man is not where he stands in moments of comfort and convenience, but where he stands at times of challenge and controversy.

—Martin Luther King, Jr.

Semantic Web technology has a lot to offer to research disciplines which are inherently multidisciplinary. The *Life Sciences* are an interesting example, spanning multiple domains ranging from pharmacy to genetics to clinical trials. This necessitates the runtime integration of different datasets of significant size. Being able to interact with these datasets as one virtual source requires technology capable of both managing *Big Linked Data* as well as successfully answering complex federated queries.

This is a challenge for RDF database architects, as they are responsible for choosing a system architecture, to handle both challenges. Unfortunately, the means to make a well-informed choice, are not readily available. The work presented in this chapter addresses this knowledge-gap. Many co-existing technologies, both from industry and from academia, are analyzed in terms of their features and performance and cost trade-offs. With this benchmarking effort we try to achieve the *ultimate measure* of the systems under test, by challenging them with Big datasets and complex federated queries.

First, we will motivate the shift in focus in this work from Pattern Mining to Data Publication in section 7.1 . We will enlist a number of requirements for a successful data publication strategy in Life Sciences. Next we will give an overview of all recent benchmarking efforts and position this chapter amongst these efforts.

In section 7.3 we describe the broader context of the SEQUEL project and the related datasets which are used for testing these database management systems. In section 7.4 we describe our approach in creating a *reproducible* and *reliable* query performance assessment and the selection procedure for the stores involved. The results sections 7.5 - 7.7 gives as an overview of the different options and their associated trade-offs when selecting the most appropriate Linked Data Infrastructure in the context of Big Data. Apart from parameters associated with the infrastructure and the performance, we also investigate the effect of different query types and whether engine performance is consistent when comparing results on artificial versus real-world benchmarks.

7.1 Publishing Life Science Data as Linked Data

Life Sciences is one of the successful application domains of semantic technology. The Life Sciences domain is interdisciplinary, which makes interlinking data sources interesting and crucial. Linked Data can alleviate the burden of data integration by explicitly incorporating the semantics in the data, using standardized vocabularies (ontologies). Each separate discipline can then publish their data, by maximally reusing existing ontologies, thus producing 5-star Linked Data.

Link with part II: BLSSpeller Motif Database The output of the BLSSpeller algorithm is different than is the case for most motif discovery algorithms. If we take Wang [54] as an example, typically a set of statistically overrepresented DNA motifs are reported in a motif discovery research paper. BLSSpeller on the other hand, provides statistics on 6.6 billion DNA patterns (selection criterion $F_{min} > 0.5$), corresponding to a size on disk of approximately 500 GB. BLSSpeller does however not imply that any of these motifs is necessarily biologically relevant. The output of this algorithm could therefore be more accurately described as a *motif database*.

How to interact with this Motif Database? A question which arose quickly after the publication of the BLSSpeller paper is how this information could be *pub-*

lished and *consumed*. The paper itself provided the means to reproduce all data by publishing the input datasets and MapReduce software[1]. This, however, still provides a major obstacle to continue this research track, as generating the motif data requires serious computational resources and the data engineering skills to run parallel software in a cloud or HPC setting.

The motif data could be published in 3 different ways, each with a different type of interaction in mind. (see Figure 2.2)

1. Integrate the information in a specialized platform for analysis and (interactive) visualization such as the PLAZA platform [52], which harbors the output of the i-ADHoRe algorithm from chapter 5.

2. Create a number of static datasets, which can serve as the input for data mining algorithms. The algorithms can then try to extract additional insights by for example clustering the motifs in the database.

3. Ingest the data in a queryable database and put this on the WWW via a Web API allowing end-users to obtain a filtered subset of the data.

The *consumption* of motif data is obstructed by:

- **The loss of semantics:** The data is not self-descriptive and confidence/conservation metrics and algorithm parameters are only defined in a research paper.

- **The challenge of data integration:** The focus of the motif discovery algorithm is on the DNA level. To fully comprehend the process of transcriptional regulation a lot of complementary information is required: Information on gene function (gene ontology analysis), the RNA (transcriptome), the methylome, the 3D structure of the DNA,... The integration of all this information requires a lot of expert knowledge and is very labor-intensive

- **The publishing bottleneck:** As mentioned previously, generating the motif database requires running a distributed MapReduce algorithm. Publishing Big(!) Data for further consumption comes with its own set of challenges on how to handle the dataset size, how to design a Web API, and how to query/slice the dataset.

[1] `http://bioinformatics.intec.ugent.be/blsspeller/`

In chapter 3, we saw that the Semantic Web stack provides solutions to all of these requirements:

- Publishing Life Sciences data as *Linked Data* takes care of the issue of **semantics**.
- If Linked Data is generated properly, its **integration** with other Life Sciences data is automatic, without any additional effort, which partially tackles the second issue.
- In the Semantic Web data is most commonly published using a SPARQL endpoint, which is a Web API that can be queried using a standardized query language called SPARQL.

Big Data and Federated Querying The Linked Open Data Cloud contains many RDF data sources related to life sciences, as shown in Figure 3.6, but running queries on top of this data comes with additional challenges:

1. The union of all datasets qualifies as Big Data and therefore puts a strain on the available technologies for querying
2. Interesting questions often combine information from multiple datasets at once, making the queries federated in nature.

These challenges show that the Life Sciences domain qualifies as a case of *Big Linked Data*, which was previously discussed in section 3.4.

Trade-offs Choosing an RDF database and system architecture requires making trade-offs:

1. Which features are essential to the system of choice, which are optional?
2. What hardware is required to achieve a certain performance?
3. What system is most suited for a specific use case?
4. What are the trade-offs when using research prototypes?

To make matters even more complicated, database vendors are continuously improving their products making it unclear when prior results become obsolete.

The goal of this chapter is to give an up-to-date view on the RDF storage solution space. By releasing scripts for deployment and post-processing of results, we provide a feasible approach to run benchmarks with own data and queries using only a limited time window of a couple of days. This work also offers a methodology to make benchmarks more reproducible and therefore the results more easily generalizable.

Research Questions

The work presented here is built around 4 research questions:

1. *How to run a query performance benchmark in a reproducible and reliable way?*

2. *What are the different options and the associated trade-offs when choosing a linked data infrastructure setup in the context of Big Linked Data? How can different setups be compared?*

3. *What is the relative influence on the measured performance of contextual factors (for example: caching) for the different RDF solutions. Is the impact similar for all solutions?*

4. *How do the RDF systems behave in a real-world setting? Can we extract insights that might be transferred and generate hypotheses to be verified in future benchmarks?*

The methodology we propose to run a scientific benchmark, is discussed in section 7.4. In the result sections 7.6 and 7.7, a subsection is dedicated to *query completeness* and *query errors* respectively, both related to benchmark reliability.

The different approaches to scaling and how these affect the query runtime performance are discussed in section 7.5. Section 7.6 addresses some of the challenges in the interpretation of the runtime results, by analyzing the impact of multi-threading, the role of caching and the run-times for different types of query templates.

The performance on the real-world dataset of Ontoforce is the subject of section 7.7.

Personal Contributions and Project Context

The research activities in this chapter were funded by VLAIO (the Agency for Innovation and Entrepreneurship in Flanders) in an R&D project titled SEQUEL[2] with Ontoforce[3], Ghent University and imec. The context of the SEQUEL project will be discussed in section 7.3.

The actual benchmarking was a shared effort by Dr. De Vocht and myself. Prof. Verborgh and Prof. Mannens were actively involved in giving scientific counseling and provided in-depth reviews for the research papers. From the side of Ontoforce, co-authors Filip Pattyn, Kenny Knecht and Hans Constandt helped defining the *industrial need* addressed by this work and created a challenging real-world data and query set.

7.2 This work versus related benchmarking efforts

Recent Benchmarking Efforts

There's an abundance of Linked Data benchmarks mainly operating on artificial datasets, the most popular ones being (chronologically) the Lehigh University Benchmark [23] (LUBM), the SPARQL performance benchmark [48] (SP^{2}Bench) and the Berlin SPARQL benchmark [5] (BSBM). For real-world data and queries the most common choice was to use the DBpedia SPARQL benchmark [33] (DBSB), which uses the DBpedia dataset and the (mostly BGP) queries extracted from the actual server logs.

Diversified Stress Testing to address vendor optimizations The shortcomings of these early benchmarks were addressed in recent work, which resulted in the Waterloo SPARQL Diversity Test Suite [2] (WatDiv). This new benchmark focuses on *diversity*, both in terms of the query properties and data properties. The first is achieved by generating queries from 20 BGP query templates, with different shapes, corresponding to different join types. The latter affects the *triple pattern selectivity* and therefore reveals the ability of RDF system's internal query planning algorithms to make the most efficient choice to resolve a query. In this work we will use WatDiv to assess the current state-of-the-art or RDF storage systems.

[2] SEQUEL: **SE**mantic Federated **QU**ery **E**ngine for **L**ife Sciences and beyond.
[3] `http://www.ontoforce.com`

Table 7.1: Overview of recent (2011-2016) benchmarking results.

Benchmark/Paper	Year	Dataset(s)	Triple Stores	Nodes x RAM	Remarks
Graux et al. [22]	2016	WatDiv1k, LUBM1k, LUBM10k	Standalone (CumulusRDF [31],...) HDFS with prep.: S2RDF [46],... HDFS no prep.: PigSPARQL [44],...	10 x 17GB	
SPB [30]	2016	SPB64M, SPB256M, SPB1B	Virtuoso, GraphDB	Virt(192 GB), Gra(64GB)	
Hernandez et al. [26]	2016	Wikidata	4store [24], Blazegraph, GraphDB, Jena TDB, Virtuoso, Neo4J, PostGreSQL	1 x 32GB	
S2RDF [46]	2016	WatDiv10M, WatDiv100M	S2RDF [46], H2RDF+ [37], Sempala [45], PigSPARQL [44], SHARD [39], Virtuoso	10 x 32GB, Virt(1 x 32GB)	
BigRDFBench [40]	2015	13 real datasets	FedX [49], SPLENDID [20], ANAPSID [1], FedX+HiBISCuS [43], SPLENDID+HiBISCuS	1 x 8GB	FedBench + 18 new queries
FEASIBLE [42]	2015	generator	Virtuoso7, Sesame, Jena TDB, OWLIM-SE	1 x 16GB	
WatDiv [2]	2014	WatDiv10M, WatDiv100M	MonetDB [7], RDF-3X [34], Virtuoso6, Virtuoso7, gStore [58], 4store	1 x 16GB	
Cudré-Mauroux et al. [11]	2013	BSBM (10, 100, , 1000M) DBPSB	4store, Hive+HBase, CumulusRDF, Couchbase, Jena+HBase [28]	2^n x 8GB $n = 0, 1, ...4$	
BioBenchmark Toyama [55]	2012	5 biological datasets (10M - 8000M) Uniprot, DDBJ,...	4store, BigData, Mulgara, Virtuoso, OWLIM-SE	1 x 64GB	5-20 queries per dataset
FedBench [47]	2011	11 endpoints with $\leq$ 50M	SPLENDID, Alibaba, Sesame	1 x 32GB	14 federated queries (7 life sciences, 7 cross-domain)

Diversity in terms of SPARQL properties Query diversity in terms of SPARQL properties is one of the features of FEASIBLE [42]. Here, the queries are selected by first converting them to normalized feature vectors and then choosing a set of mutually distant queries. Also the Semantic Publishing Benchmark [30] (SPB) provides more complex query workloads with nested queries. In SPB all SPARQL 1.0 operators are present.

Synthetic versus Real-World benchmarks: Apples and Oranges? A recurring criticism on synthetic benchmarks is that they have very little in common with real application domains [15], therefore it is not possible to generalize benchmark results of RDF databases on artificial data to real-world use cases.

Benchmarks with Life Sciences data If we look specifically to the Life Sciences domain, BioBenchmark Toyama 2012 [55] sheds light on the capabilities of typical single-node RDF storage solutions. They evaluated 5 triple stores on 5 biological

datasets (Cell Cycle Ontology [4], Allie [56], PDBj [29], UniProt [9] and DDBJ [51]), ranging from 10 million to 8 billion triples.

Multi-node benchmarks All benchmarks mentioned so far focus on single node RDF databases. FedBench [47] is a system to test query federators. They evaluate 3 federated systems using 14 real-world federated queries, 7 from the Life Sciences domain. More recent work, BigRDFBench [40], increases the number of datasets from 11 to 13 and adds 18 new federated queries. Instead of just focusing on query runtime other performance metrics are taken into account such as source selection and *query correctness*. An alternative heuristic approach for automatically generating federated queries is the SPARQL Linked Open Data Query Generator [21] (SPLODGE).

Mapping SPARQL workloads on other types of NoSQL systems Most benchmarking efforts reported so far focus on the performance of native RDF systems. A first generalization comes by adding other graph database systems and relational databases, as in the WikiData benchmarking effort [26], where Neo4J and PostgreSQL were added. A second generalization comes by mapping SPARQL workloads on NoSQL and Hadoop-based systems. Graux [22] compared 3 different types of systems: (i) Standalone NoSQL based approaches such as CumulusRDF [31] (translates to Cassandra). (ii) HDFS-based (Hadoop Distributed File system [19]) approaches with a data preparation phase such as S2RDF [46]. (iii) HDFS-based approaches which natively store RDF such as PigSPARQL [44]. This work can be viewed as an update of an earlier NoSQL for RDF benchmarking effort by Cudré-Mauroux [11]. In the S2RDF research paper [46], a comparison is made with other HDFS-based approaches and a single server instance of Virtuoso.

Current Efforts in H2020 projects The current difficulty in selecting and evaluating RDF systems is also being addressed in two European H2020 projects: LDBC [3] and HOBBIT [35]. Within LDBC a number of RDF benchmarks were developed [6], one benchmark is based on social network data [16] and SPB [30] based on a data publishing case with BBC. In the HOBBIT project a platform is being built to offer industry a unified approach for running benchmarks related to their actual workloads.

Positioning this work

In Table 7.1 we provide an overview of the most recent benchmarking results together with information on their time of release, the datasets used, the systems tested, and the hardware setup. Our work distinguishes itself from other effort as follows:

Up-to-date view Specifically for the Life Sciences domain BioBenchmark Toyama 2012 is the most recent report for single-node setups.

Scalability We both study scalability in terms of dataset sizes (WatDiv), but also in terms of the size of the distributed setup (horizontal scalability) and in terms of memory resources (vertical scalability).

Broad set of query types Where the WatDiv runs are diverse in the space of BGP queries, the queries of Ontoforce are complex, rich in SPARQL keywords, subqueries are common and a large fraction consists of non-conjunctive queries, which are typically very challenging [38].

Query Correctness Just like BigRDFBench we explicitly verify query correctness before turning to runtime comparisons and demonstrate its necessity for challenging queries.

Objective and exhaustive By considering different hardware and configuration setups our work becomes more objective. As an example the S2RDF paper compares Hadoop-based systems with Virtuoso and concludes a similar performance, but does not take into account that (as will be shown later), performance does not drop when adding multiple clients, thereby increasing Virtuoso's ETL throughput by an order of magnitude.

Multi-setup This work compares single and multi-node setups, federated querying, and compression by using benchmark cost as a unification parameter.

Query-mix size Whereas many benchmarks have a limited query-set, both the WatDiv and Ontoforce benchmark used in this evaluation can be considered stress tests with respectively 400 and 1,223 queries.

Flexibility Any system can be tested with our approach, the only requirement is support for the SPARQL protocol. Because of this we can for example also test the Triple Pattern Fragments (TPF) system, since the TPF client can be run as an http-server.

7.3 SEQUEL: Context and Datasets

This research was conducted in collaboration with Ontoforce, a company specialized in semantic search solutions for Life Sciences data.

Ontoforce has designed a semantic search platform DISQOVER[4] which integrates over 110 Life Sciences data sources. Examples of these datasets are PubMed, ClinicalTrials.gov, NCBI Gene, National Drug Code, MedDRA, DrugBank, MeSH, etc.

Ontoforce has built its own central ontology to integrate these different datasets. The DISQOVER UI enables the interactive exploration of these Linked Datasets. Actions in their search interface trigger *federated queries* in the back-end of their product. These queries are related to *faceted browsing*. A typical interaction with DISQOVER is initiated by a keyword search as can be seen in Figure 7.1 which shows the data sources that contain information about the keyword query, in this example 'Ibuprofen'.

In the back-end this faceted browsing corresponds to starting from 'more general' and moving towards 'very specific' queries. The specificity is added by using facet filters, also in the corresponding SPARQL queries this corresponds to appending FILTER statements to the initial general queries. An examples of facets in different categories is shown in Figure 7.2 further building upon the search in the previous figure.

The *goal* of the SEQUEL project was to find or design a system which could:

- resolve federated queries including **distributed** joins.
- query **live SPARQL endpoints** containing life science resources.
- resolve these queries in **responsive** fashion with low latencies.

[4] `https://www.disqover.com`

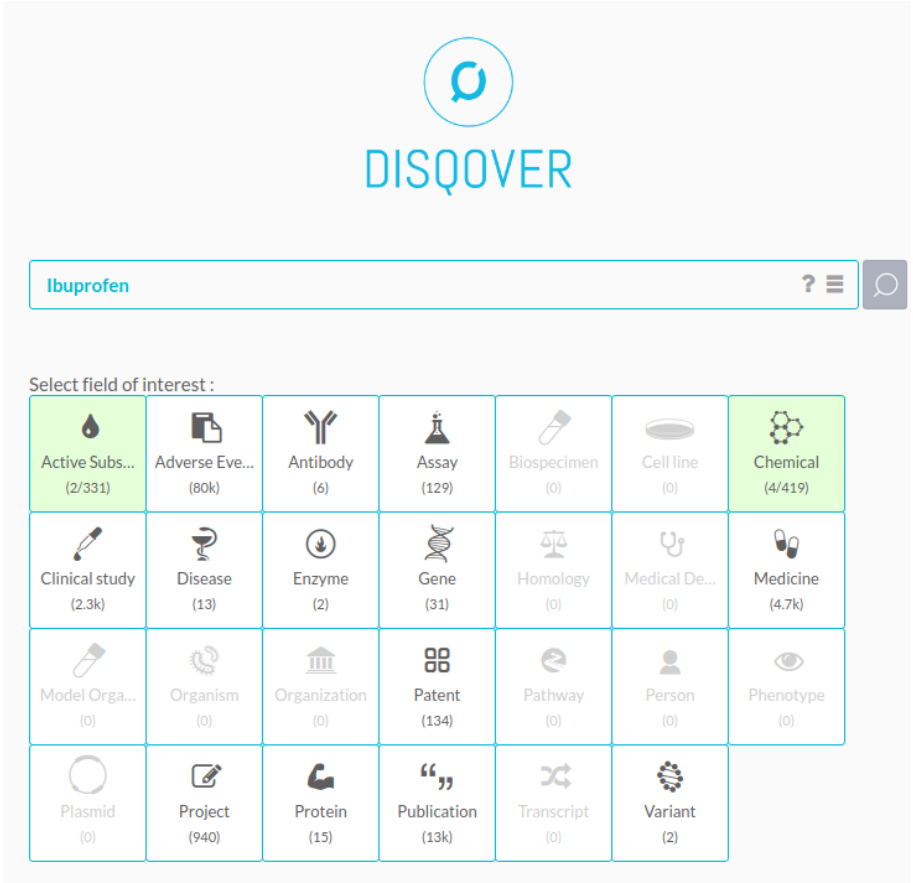


Figure 7.1: A keyword query for 'Ibuprofen' in the DISQOVER UI results in an overview of the data sources that contain information related to this keyword. These sources can be selected for further narrowing down the search.

Initial Benchmarking Efforts In order to have a 'smooth user experience' actions in the front-end should result in responses in less than 100 ms.

To assess the feasibility of this requirement, we set up a test using FedBench [47]. There we compared 3 approaches for federated querying. The test consisted of 9 federated queries, described in Table 7.4 on page 184, where the answer required information from both ChEMBL and Drugbank:

- ChEMBL: A manually curated chemical database of bioactive molecules with drug-like properties.

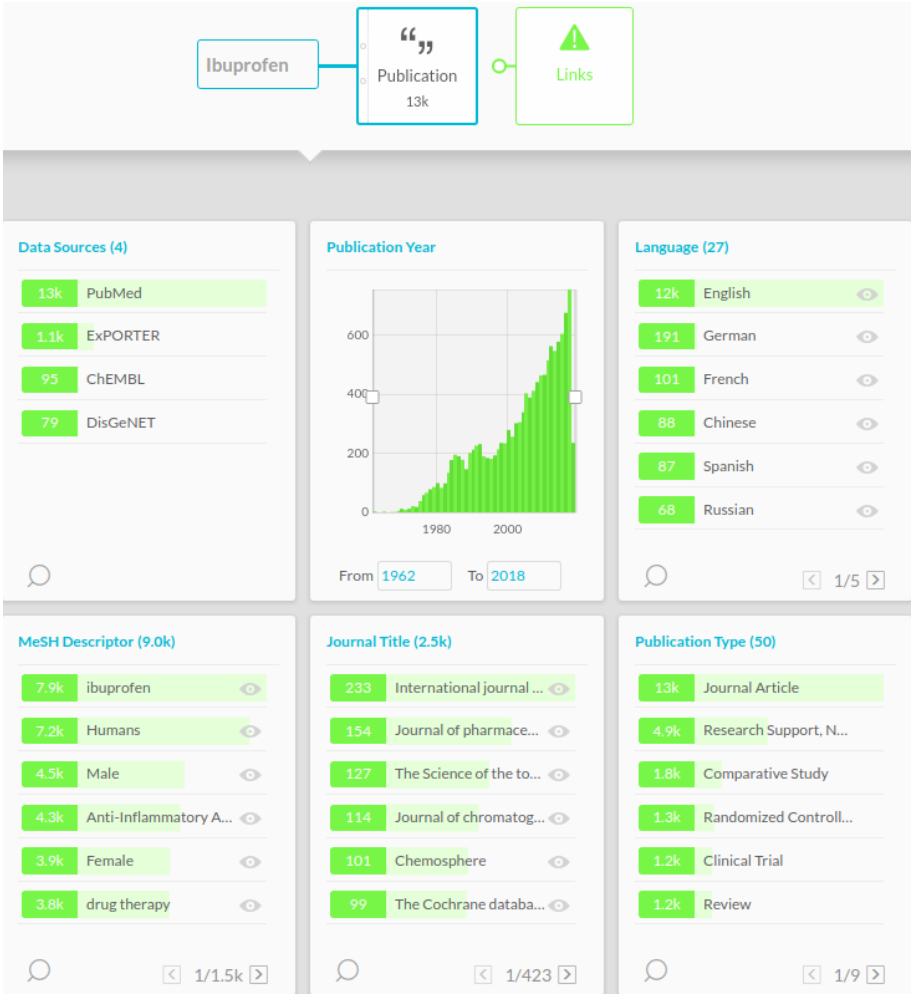


Figure 7.2: Facets related to the search for Ibuprofen, after selecting 'Publication' in Figure 7.1. The search for publication can be further narrowed with facets per data source, publication year, language,...

- DrugBank: The database contains drug entries, linked to this are protein sequences which are targeted by these drugs.

Three different federation frameworks were chosen to test:

- Distributed SPARQL on Sesame [8], which has no optimization of any kind.
- FedX [49] is known as a solution with state of the art performance.
- IDLab DARQ [10], which is an internally developed optimization of Distributed ARQ (DARQ) in Jena [27].

With the two datasets hosted by 2 separate Virtuoso (open source v6) instances the median execution times are shown in Figure 7.3

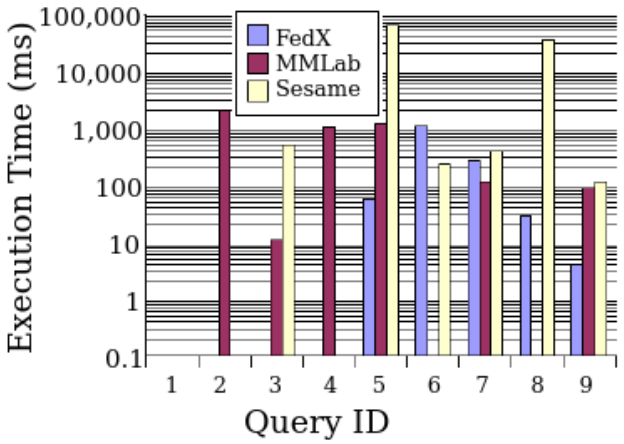


Figure 7.3: Execution times for federated queries on chEMBL and DrugBank hosted on separate SPARQL endpoints. Some queries for which the runtime is shown are however incorrect!

The *Sesame* engine solves 5 queries correctly, fails on 3 due to timeout ($\geq 240s$) and 1 query is incorrect. *FedX* manages to solve 5 queries, while the other 4 queries fail. The query failures are caused by a lack of support for some SPARQL 1.1 operators such as *nested selections*. The *DARQ* engine has execution times for 6 out of 9 queries, with 1 query resulting in a timeout and 2 resulting in a failure. Unfortunately 4 queries also led to the wrong result. The issues with a *lack of query correctness* highlight a weakness in query solutions developed in the context of research papers, therefore in what follows we will focus on *more mature vendor-backed*

RDF database solutions. The latter is defined by requiring full support for SPARQL 1.1, a mature and actively maintained codebase, sufficient community adoption,...

Apart from the other issues also the query execution times often *exceed the 100ms requirement*. If we focus on FedX alone, in Figure 7.4 the trend of having more endpoints/datasets seems to deteriorate the query execution times with at least one order of magnitude.

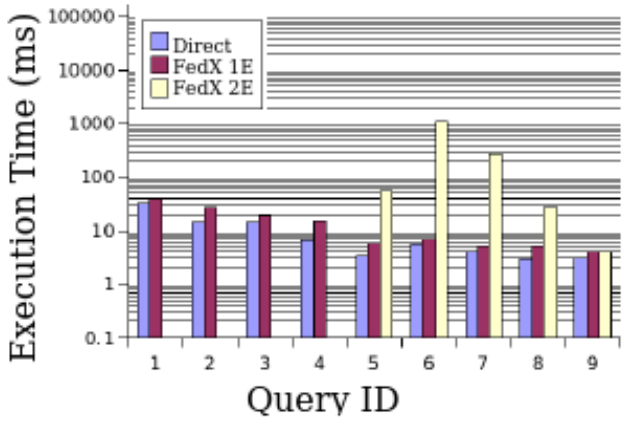


Figure 7.4: Increasing the number of endpoints for FedX generally leads to an increase in runtime of an order of magnitude. This is a problematic trend since the number of sources $\mapsto$ 110. Direct corresponds to a single SPARQL endpoint, thus sidestepping FedX.

The conclusion drawn from this initial effort, was that the project goal, to improve upon the DARQ engine, was not the most promising route. Therefore, an alternative approach was chosen, where the triple store is no longer used in the back-end, but in the ETL phase. During this preparatory phase queries are preprocessed and the data is aggregated, in well chosen linked documents, which are indexed by Apache Solr [5].

The ensuing benchmark efforts therefore focus on the simplified architecture shown in Figure 7.5: we try to determine the best approach to run the ETL pipeline, using an artificial dataset. In a later stadium the faceted queries, generated in the DISQOVER UI and extracted from the query logs, are used to evaluate the solutions in a real-world scenario.

[5] `http://lucene.apache.org/solr/`

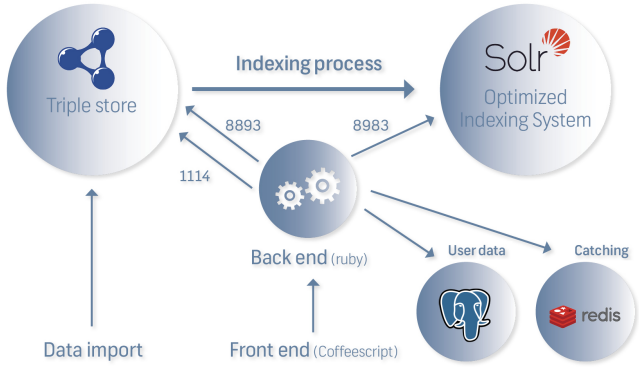


Figure 7.5: Proposed architecture where RDF database (Virtuoso) is used only in an ETL pipeline to create Linked Documents, indexed by SOLR.

A major difference with this setup is that now the scope can be broadened to not only include solutions for federated querying since the data can now be centralized. With this offline setup other approaches come into scope leading to a *high-dimensional benchmark parameter space*, as discussed in section 7.4. Additional control is gained in terms of the database software, performance tuning, the choice of hardware, data partitioning approaches,...

This immediately led to the blueprint for two research papers. In the first work [14], we evaluated 4 RDF databases on WatDiv [2], with 3 different dataset sizes: 10M (10 million), 100M and 1000M triples. The specifics about WatDiv will be discussed in the next section on benchmark datasets. In this work, the so-called *Vendor* systems were run as-is, without any configuration. This initial work served as an inspiration to a set of additional challenges:

1. Will the results improve given better single-node hardware?
2. How will these systems behave when configured optimally?

The second research paper [13] focused on distributed approaches, both homogeneous distributed systems as heterogeneous approaches. A real-world Life Sciences dataset was used, originating from the back-end logs of Ontoforce's DISQOVER product. In this work, also a number of research prototypes were tested, which we will label *SemWeb* systems.

This effort very clearly demonstrated, the limitations of current SPARQL solutions. We are also convinced that benchmark analysis should also focus on benchmarks in

which failures are common, up until the point that they terminate the RDF database system.

Follow-up research, given in the result sections, tries to provide answers to these additional questions.

Benchmark Datasets and Queries

We used both artificial and real-world datasets to assess the performance of the most promising RDF solutions. For the artificial datasets we made use of the WatDiv [2] benchmark generator. Ontoforce provided us with a proprietary dataset generated from the interactions with their DISQOVER interface.

WatDiv: Diversified Stress Testing

WatDiv was chosen due to its promise of offering a benchmark which is *diverse* both in terms of queries and data properties. In the research paper [2] it was also shown that competing benchmarks lack in diversity.

Data Diversity Diversity at the level of the data is achieved by having diversity in the *result cardinality* per query and the *triple pattern selectivity* (tp-selectivity). In Figure 7.6 (taken from WatDiv site) the **L1** query is shown with stats about these data-driven properties.

	?v0	?v2	?v3
?v0 wsdbm:subscribes %v1%	15	n/a	n/a
?v2 sorg:caption ?v3	n/a	2502	2502
?v0 wsdbm:likes ?v2	23922	23800	n/a
overall	2	2	2

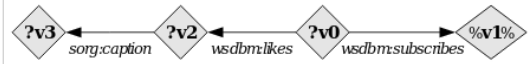


Figure 7.6: **L1** query with details about the number of results (overall) and tp-selectivity for WatDiv10M. A successful query planner will prefer the first tp to the third since ?v0 has only 15 bindings instead of 23922. The possibilities ?v2 might be reduced significantly in combination with ?v0. Intuitively the order should therefore be: $tp_1 \mapsto tp_3 \mapsto tp_2$

With the WatDiv benchmark generator we created datasets of 10, 100 and 1000M (million) triples, which correspond to scaling factors S = $10^2, 10^3, 10^4$. Global statistics about the triples in WatDiv are shown in Table 7.2.

WatDiv datasets are also used in federated setups. In these setups the dataset is *partitioned* using subject hash partitioning [25, 57] which led to 3 equally-sized

datasets. Note that this partitioning scheme benefits star-shaped queries as they can be resolved without inter-node communication.

Property	Value	Property	Value
Number of distinct subjects	$5.6k \cdot S$	Number of triples	$105k \cdot S$
Number of distinct predicates	85	Number of graphs	1
Number of distinct objects	$13.3k \cdot S$	Compressed (nt.bz2) dataset	$0.6S$ MB
Number of distinct classes	16		

Table 7.2: The WatDiv dataset in numbers, with S a scale factor, corresponding with different dataset sizes: $S = 100$ for WatDiv10M

Query Diversity Diversity in terms of queries is implemented as having queries rich in structural features. The queries are all basic graph patterns (BGPs). Therefore, WatDiv reveals the ability of today's triple stores to handle different types of complex join operations.

The following structural features are considered:

- **Triple Pattern Count:** The number of edges in the BGP.
- **Join Vertex Count:** The number of internal vertices in the BGP.
- **Average Join Vertex Degree:** The average degree of all vertices in the BGP.
- **Join Vertex Types:** Two triple edges (spo) can be joined on the subjects (SS+), on the objects (OO+) or mixed (SO+).

WatDiv comes with a set of 20 query templates diverse in terms of these features, see also Figure 7.7 (adapted from WatDiv site[6]):

- L: Linear chains (**L1** - **L3**)
- S: Star-shaped queries with one central node (**S1** - **S7**)
- F: Snowflake queries are a combination of S queries (**F1** - **F5**)
- C: Combinations of the above (**C1** - **C3**)

Per query template we generated 20 queries, corresponding to 400 queries.

[6] `http://dsg.uwaterloo.ca/watdiv/basic-testing.shtml`

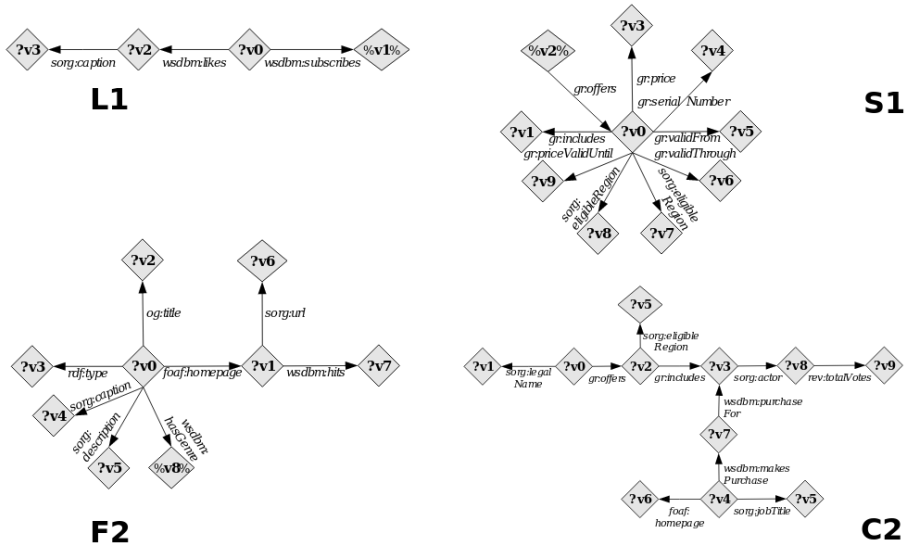


Figure 7.7: 4 template types for WatDiv (%v% are placeholders for constant URIs):
L1: Linear chains have 3-4 vertices, 1-2 join vertices, an average join degree of 1-2 and SS+ or SO+ join types.
S1: Star-queries have one central high-degree join vertex connected with 4-9 other boundary-vertices. Average join degree is 2, and SS+ or SO+ join types.
F2: Snowflake queries have 6-10 vertices, 2 high-degree join vertices, an average join degree of 1-2 and SS+ or SO+ join types.
C2: Combinations of previous templates, chains have 7-10 vertices, 1-3 join vertices, an average join degree of 1-2 and all join types, including OO+.

Ontoforce: Real-world Life Sciences data and queries

Property	Value	Property	Value
Number of distinct subjects	0.137B	Number of triples	2.4B
Number of distinct predicates	1,782	Number of graphs	107
Number of distinct objects	0.287B	Compressed (nq.gz) dataset	25GB
Number of distinct classes	2,434		

Table 7.3: The combined statistics show the high demands of the datasets used by the DISQOVER platform.

Big Multi-graph Life Sciences dataset One real-world proprietary dataset was provided by Ontoforce. The dataset consists of 107 graphs with a total of 2.4 billion

triples. The majority of the graphs are cleansed versions of Linked Open Data in the Life Sciences section of the LOD cloud. For every graph, an additional graph is present containing the inferred triples, integrating everything in Ontoforce's central ontology. The five largest graphs make up approximately 80% of the dataset size. These graphs correspond to PubMed, ChEMBL, NCBI-Gene, DisGeNET and EPO, with the PubMed already making up 60% of the data.

Table 7.3 contains global statistics about the benchmark dataset. The dataset is only a fraction of the complete dataset used in DISQOVER which contains over 8 billion triples. It contains the graphs which are most relevant to the queries in the operation mix. As shown in our prior work with WatDiv [14], datasets exceeding 1 billion triples can pose serious challenges, to even state-of-the-art enterprise RDF databases given modest hardware.

Since the dataset is under nondisclosure we provide some additional statistics in Figure 7.8 to shed some light on the data and help researchers create artificial benchmarks with similar dataset properties.

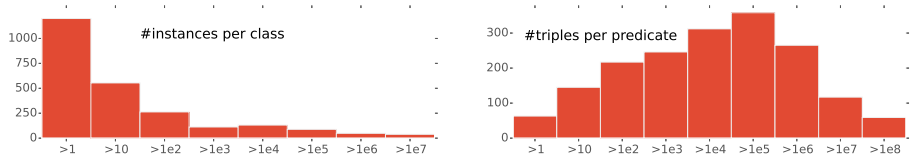


Figure 7.8: The number of instances per class (left) follows a long tail distribution: the bulk of 2,000 classes has 1–100 instances, while the long tail contains 50 classes with 500K instances each. The median of the predicate distribution (right) occurs at 5,000 triples, while the outliers reveal that 50 predicates have over 10M triples each.

Queries rich in SPARQL features The querymix provided by Ontoforce was extracted from the user logs of the DISQOVER search interface.

The benchmark querymix consists of 1,223 queries which are both complex and diverse in term of SPARQL features and are deeply nested.

While WatDiv queries are all BGPs, the DISQOVER queries are interactive federated queries associated with faceted browsing [18, 36].

Due to the automated way of generating queries, the formulation of the queries is not optimized in terms of performance [32]. From the point of view of Ontoforce this optimization is considered the responsibility of the triple store.

Table 7.4: Examples of Federated Queries for Drugbank and chEMBL used in the initial benchmarking effort. COUNT operator occurs in Q1, Q4, Q5 and Q8. Hints for the UNION operator are the 'OR' and the 'any of' in Q1 and Q2. Q6 and Q7 require a SORT operator to get the 'top' results. Q4 requires a GROUPBY operator for the different development phases.

Query	Description
Q1	**Count** the total number of drug-drug interactions for drug "Butalbital" **OR** "Lithium" in Drugbank
Q2	Get the relations to targets where **any of the drugs** "Dihydroergotamine" and "Rizatriptan" are antagonist of. We present chEMBL drugs we got from a previous query, while the relationships are in DrugBank.
Q3	Get specific properties for the drug "Butalbital" to populate its summary instance view, from both DrugBank and chEMBL.
Q4	Get **the number of drugs per development phase** having "migraine" their description, for manufacturer "Sandoz inc". Phases come from chEMBL, manufacturers come from DrugBank.
Q5	**Count** the number of distinct drugs that have "migraine" in their description, for certain manufacturers, from DrugBank. Instances with different URIs that should be considered the same, are counted twice.
Q6	**Get the top 15** manufacturers of drugs that have "migraine" in their description from DrugBank.
Q7	**Get the top ATC** (level 1) codes for drugs having "migraine" in their description, for certain manufacturers, from DrugBank.
Q8	**Count** the number of distinct drugs that have "migraine" in their description, from DrugBank and chEMBL. Instances with different URIs that should be considered the same, are counted twice.
Q9	Get all URIs used to represent the drug "Butalbital".

The queries are automatically built by the system from general queries where - while browsing - additional FILTER statements are added. Aggregation operators and filter operations are therefore predominant, a large fraction of queries is also non-conjunctive (UNION and OPTIONAL), making them even more challenging [38]. Queries with over 10 triple patterns are common and more specifically unbound triples occur often, the actual binding occurs in the additional FILTER statements. Half of the queries are COUNT DISTINCT queries and these are also the most time consuming to resolve. Table 7.4 gives query descriptions for the federated queries used in the initial benchmarking effort. These queries are of the same nature as the other faceted browsing queries.

To provide the reader with more systematic insights in the queries we used SPARQL.js parser[7], which converts SPARQL queries into JSON objects. This JSON structure is then used to generate a feature vector per query.

We distinguish between features related to the complexity of the query structure and features which correspond to SPARQL keyword counts.

> **Example feature vector of Q4:** Triple Patterns(11), nested select queries(3), query file size (< 1kb), operators: OPTIONAL(1), GROUP(1), ORDER(1), COUNT(1), UNION(1), FILTER(7), FILTER IN(2).

In Figure 7.9, a series of features and their occurrence distribution is shown, which shed further light on the complexity of the SPARQL patterns.

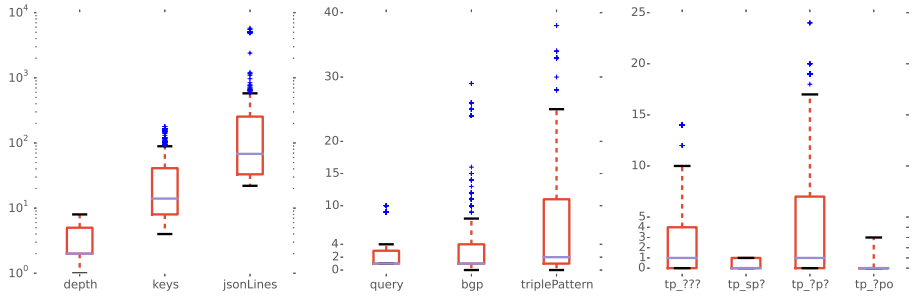


Figure 7.9: The range of depths (left) indicate many queries are nested, jsonLines shows that some queries are very long (mostly caused by FILTER IN). 25% of the queries contain more than 10 triple patterns (center), but most of these are completely unbound (???), or only have a predicate (?p?) (right).

[7] https://www.npmjs.com/package/sparqljs

Three panels of keywords which are related to:

1. properties of the JSON tree representation such as the number of levels (depth), the number of nodes (keys) and the length of the file (jsonLines);

2. properties of the query graph structure such as the amount of queries, BGPs and the total amount of triple patterns;

3. the type of triple patterns.

The large amount of queries with over 10 triple patterns is noteworthy, while the WatDiv query templates contain 3 up to 10 patterns maximally. The prevalence of unbound triples reveals how the DISQOVER queries are built: starting from general queries where additional selectivity is introduced by ad hoc introduction of FILTER statements. Most of these FILTER ?x = <...> queries can be manually removed by replacing the corresponding ?x variable in the triple patterns. Other FILTER operations contain an IN operator followed by a long series of possible values for a variable, which could be rewritten as complex unions. Half of the queries are COUNT DISTINCT queries, furthermore most keywords are present, their effect on query run-times is shown in Figure 7.10. This Figure will be revisited in the results section.

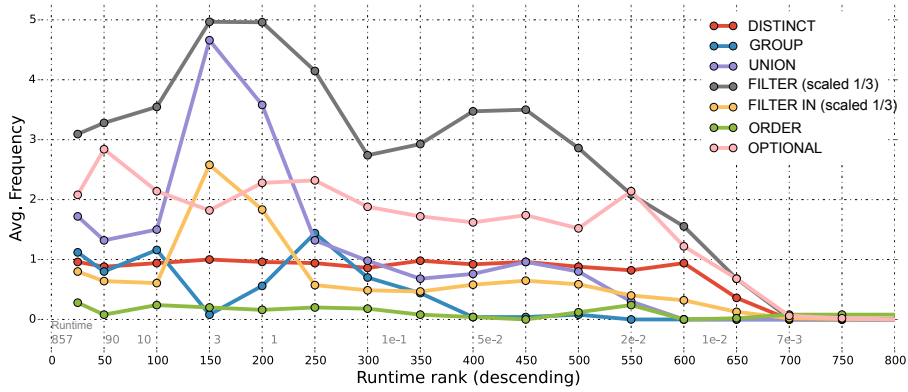


Figure 7.10: Occurrence of SPARQL keywords per query (averaged). The extremum for query numbers 100 to 150 indicates that queries with a lot of FILTERs and UNIONs are among the most challenging ones, yet the left hand side contains queries with a higher number of OPTIONAL and GROUP keywords and the result set sorted.

7.4 Benchmark Approach

In this benchmark we would like to discover trends when modifying certain aspects of the benchmark setup. In this section we define a *benchmark space*. For every dimension in this space we try, to at least test two possible values. Performance is not always the first concern in a system architecture. In section 7.4, we describe an approach using a Feature Matrix, in which weights can be assigned to certain properties of a system, in order to make a ranking of the different systems, given a use case. Section 7.4 gives a detailed explanation, about our attempt at making the benchmark itself more easily reproducible and comparable with other work.

Benchmark Space Exploration

Assessing the performance of an RDF system with a given benchmark starts with the identification of the set of parameters its results depend on. The actual outcome is a function of (at least) the following dimensions, for which we test multiple values:

1. **The choice of database engine:** We assess 7 different systems, 4 *Vendors* and 3 *SemWeb* systems.
2. **The server hardware—especially memory:** We distinguish between 32GB and 64GB of RAM on the server.
3. **The size of the (optionally) distributed system:** We run tests for single and 3-node setups when supported by the RDF database. Federated systems consists of $N+1$ nodes, with N the number of slaves (1 or 3) nodes, and 1 federator node. To clarify: $N=3$ thus corresponds to 3 instances for *Vendor* systems, while $N=3$ for federated setups requires $3+1$ instances. The choice for $N=3$ is related to the fact that for one of the systems only a 3-node configuration is available.
4. **The query properties:** The WatDiv benchmark query-set contains BGP queries, while the Ontoforce dataset consists mainly of complex aggregation-based queries.
5. **The number of dataset triples:** We run 3 datasets of WatDiv, with 10 million, 100 million, and 1 billion triples. The Ontoforce dataset contains 2.4 billion triples.

Table 7.5: Overview of benchmarks run in this study.

Systems	Setup	WatDiv			Onto-force
		10M	100M	1000M	
Vendors (4)	32	✓	✓	✓	
	64			✓	
	64/Opt			✓	✓
Multi-Node (2)	3 x 32			✓	
	3 x 64/Opt				✓
Semantic Web (3)	64		✓	✓	✓
	3 x 64		✓	✓	✓

6. **The way in which the RDF system is configured:** We used the recommended configuration in the store's documentation as the *Default* configuration and sent out a request for information to the vendors to achieve an *Optimized* setup for WatDiv1000M.

7. **The state of the system when the query is launched:** We distinguish between a single-threaded warm-up run and a multi-threaded stress test (5 clients). We also investigate whether caching effects play a role in the runtime behavior.

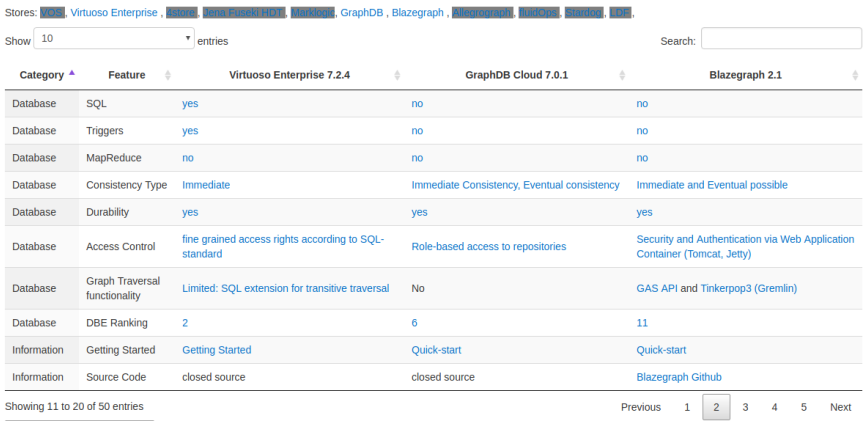
Testing every possible combination of parameters is very time and resource consuming and not necessarily the most informative. Therefore, we opted for a greedy exploration of this space, consisting of 51 2-phase benchmarks (incl. re-runs), each with a warm-up and a consecutive stress test. Table 7.5 gives an overview of the benchmarks we performed.

Store Preselection: Feature matrix

We created a Feature Matrix, as shown in Figure 7.11, and evaluated a number of stores on a subset of those features (a similar approach as in Stegmaier [50]) to make a preselection of RDF engines we consider for in-depth analysis. We combined two ideas to create a Feature Matrix, to simplify the RDF store selection process:

- We consulted the DB-Engines ranking[8], which orders database systems ac-

[8] http://db-engines.com/en/ranking



Stores: VOS, Virtuoso Enterprise , 4store , Jena Fuseki HDT, Marklogic, GraphDB , Blazegraph , Allegrograph , BlitzDps , Stardog , CDF ,

Show 10 entries Search:

Category	Feature	Virtuoso Enterprise 7.2.4	GraphDB Cloud 7.0.1	Blazegraph 2.1
Database	SQL	yes	no	no
Database	Triggers	yes	no	no
Database	MapReduce	no	no	no
Database	Consistency Type	Immediate	Immediate Consistency, Eventual consistency	Immediate and Eventual possible
Database	Durability	yes	yes	yes
Database	Access Control	fine grained access rights according to SQL-standard	Role-based access to repositories	Security and Authentication via Web Application Container (Tomcat, Jetty)
Database	Graph Traversal functionality	Limited: SQL extension for transitive traversal	No	GAS API and Tinkerpop3 (Gremlin)
Database	DBE Ranking	2	6	11
Information	Getting Started	Getting Started	Quick-start	Quick-start
Information	Source Code	closed source	closed source	Blazegraph Github

Showing 11 to 20 of 50 entries Previous 1 2 3 4 5 Next

Figure 7.11: Interactive Feature Matrix to explore and compare the properties of the different RDF storage solutions.

cording to their data model and online popularity, to explore the currently common RDF-engines. The latter is a non-disclosed formula that measures popularity by combining online mentions on (social) platforms such as StackOverflow, Twitter, and LinkedIn. DB-Engines also supports comparing multiple features of different systems.

- WikiData selected the most appropriate RDF store to host their data by having experts assign weights to desired features[9]. These weights allowed them to calculate a score per data store and rank the different systems.

We made a broad selection of suitable features specific for RDF engines to allow multi-way comparisons. Ranking the engines is made possible by assigning weights to a set of features. The features are grouped into a number of categories, to obtain a more in-depth insight in the scoring process. The matrix is online[10], and end-users can freely download and/or extend it, change the weights, and update the scores when vendors upgrade their product. To back the scoring, we added a layer of trust to the information by always linking to the source of this information.

The criteria for selection of the *Vendor* systems in this work are closely related to the goal of benchmark space exploration and the requirements put forward by Ontoforce. The enterprise needs are met by selecting systems that offer enterprise

[9] https://phabricator.wikimedia.org/T90101
[10] http://users.elis.ugent.be/~drdwitte/featurematrix.html

support and are fully SPARQL 1.1 compliant. The benchmark space exploration requires a certain flexibility: we prefer systems with a machine image, or a maintained docker image, which put no restrictions on the amount of triples that can be ingested and that can work as a multi-node system. The application of the above selection criteria led to 4 *Vendor* systems. Later on, we added 3 additional *SemWeb* systems with unique approaches to handling RDF data: HDT [17], which is a queryable compression format, FedX [49] often included in benchmarks for federated querying, and Triple Pattern Fragments [53] as a first implementation of the Linked Data Fragments concept.

The comparison with these *SemWeb* systems was an essential part of the research collaboration with Ontoforce, as their initial goal was to build their DISQOVER search interface on top of a federated querying system. The advantage of the latter, is that their interface would then provide a live view on a continuously updating Life Sciences Linked Data cloud, removing the need for an ETL process.

All selected stores are shown in Table 7.6 together with their shorthand notation (prefix).

Table 7.6: List of the tested systems and their acronyms. The first four are enterprise systems.

System	Shorthand
Blazegraph 2.1.2	**Bla**
Undisclosed Enterprise Store	**ES**
GraphDB 7.0.1	**Gra**
Virtuoso 7.2.42	**Vir**
FluidOps[11] (with FedX 3.1.2 [49])	**Flu**
HDT-Fuseki 4.0.0[12]: Jena Fuseki to query HDT	**Fus**
Triple Pattern Fragments: Server.js 2.2[13], Client.js 2.0[14]	**TPF**

A quick and reusable benchmarking scheme

To make the benchmarks fully *reproducible*, we pay explicit attention to the hardware setup and the database configurations.

We offer a reusable infrastructure which consists of a number of well-maintained components for deployment, to allow the end-user to test a triple store. We also release our post-processing scripts and query event data publicly, so others can

reproduce the analysis of the system performance exactly as described in this paper. The Ontoforce benchmark cannot be reproduced by external parties due to the dataset being proprietary. The queries have however been released.

Reproducible hardware

The choice of hardware in benchmarks is often related to the availability of systems in a research group's data center We opted to use instances from a cloud provider to make the choice as generic as possible. We used three different types of servers on the Elastic Compute Cloud (EC2) of Amazon Web Services[15] (AWS), shown in Table 7.7.

Table 7.7: Instance types used in benchmarks and their purpose.

Instance Type	vCPUs (no.)	RAM (GB)	Goal
`r3.xlarge`	4	30	Original Choice
`r3.2xlarge`	8	61	Current Reference
`c3.2xlarge`	8	15	Benchmarker

An additional advantage of this approach is that the benchmark cost can be explicitly calculated. Using cost as a metric allows the comparison of benchmarks with different setups. Also the cost of certain preprocessing steps such as bulk loading or compression can be included in the comparison.

Reproducible installations and configurations

A very important and often not reported factor in a store's performance, is the way it was installed and configured.

A reproducible installation strategy is obtained by using Amazon Machine Images (AMIs) offered by the system vendors on the AWS Marketplace[16]. When no AMI is available we turned to well-maintained Docker images on Docker Hub[17]. The used AMIs come with a *Pay-As-You-Go* (PAGO) license, i.e., a license cost per hour of usage which also depends on the choice of hardware instances. An overview of the installation approaches followed for the different systems under test:

[16] `https://aws.amazon.com/marketplace/`
[17] `https://hub.docker.com/`

- PAGO AMIs: Virtuoso[18], GraphDB[19], ES
- Docker Hub: TPF server[20], Virtuoso Open Source[21], HDT-tools[22]
- Self-provided Docker images: Blazegraph[23], HDT-Fuseki[24]
- Manual installation: FluidOps was installed manually with FedX [41] 3.1.2 (provided by FluidOps) and Virtuoso Adapter plug-in (required since Virtuoso no longer supports ASK queries)

Our initial results with the 4 *Vendor* systems showed great differences in runtime performance [14]. After consulting with the database vendors, it turned out that this can be attributed to our choice of running the systems as-is, which we coin the *None* configuration. We decided to re-run these benchmarks using two strictly defined configurations: *Default* and *Optimized*. All configuration options are defined in Table 7.8.

The *Optimized* configuration was obtained after sending out a Request For Information (RFI) to the commercial vendors involved. The RFI asked them to provide us with scripts or configuration files to achieve optimal performance on the WatDiv1000M benchmark. GraphDB, Virtuoso, and Blazegraph responded positively to this request. A fourth commercial vendor, ES, did not respond to our RFI. Note that this configuration is not necessarily an optimal match for the real-world benchmark, as the data and queries were not shared with the vendors.

Reusable Benchmark Components

The SPARQL Query Benchmarker software[25] is a mature SPARQL-over-HTTP benchmarking tool which is highly customizable. We ran the software in *benchmark* mode where it can operate given a SPARQL endpoint URI and a list of SPARQL query files. The software was run with a timeout parameter of 300s for the WatDiv benchmarks and 1200s for the Ontoforce benchmark and with 1 single-threaded warm-up run

[18] https://aws.amazon.com/marketplace/pp/B011VMCZ8K
[19] https://aws.amazon.com/marketplace/pp/B00OM7VXGW?qid=1487779358935&sr=0-1&ref_=srh_res_product_title
[20] https://hub.docker.com/r/linkeddatafragments/server.js/
[21] https://hub.docker.com/r/tenforce/virtuoso/
[22] https://hub.docker.com/r/rfdhdt/hdt-cpp/builds/
[23] https://github.com/laurensdv/docker-blazegraph/tree/master/2.1.2
[24] https://github.com/drdwitte/rdfbenchmarking/tree/master/setup/endpoints/FusekiHDT
[25] https://sourceforge.net/projects/sparql-query-bm/

Table 7.8: Different configuration choices defined in this benchmark

Name	Description
None	Results reported in initial work were obtained modifying none of the preset configurations.
Default	Applying the recommended settings from the vendor documentation: mostly taking into account the available server memory and the dataset size.
Optimized	Settings provided by the vendors, in response to our RFI.

and a multi-threaded (5 threads) stress test run where 5 clients each execute a full querymix independently and in randomized order. The choice for timeout parameters is related to practical considerations:

- Initial tests revealed that the WatDiv timeout is sufficient for most queries to complete.
- The Ontoforce benchmark timeout was instated to keep the total benchmark execution time within affordable boundaries.

When the benchmark successfully terminates, a CSV-file is generated containing the summary results per query: median runtime, median response time,... In our initial benchmarks [14] this CSV-file was used, but with the Ontoforce dataset several issues surfaced:

1. The summary results (number of results per query and query run-times) are not correct in benchmarks where many problems arise. For example, in the calculation of the average runtime, results where the query was unsuccessfully resolved are also taken into account for the calculation of the average. It also makes it hard to verify the number of results per query. For example, a query with 10 results which is executed twice and of which one execution fails, is reported as having 5 results .
2. If the benchmarker software fails the CSV-file is not generated and the results are lost.
3. A posteriori it is not possible to verify if a query was solved correctly.

4. While the CSV-file contains useful results, it is still a summarization of the results whereby much information about the flow of the benchmarking process is lost.

These issues are however all confined to the process of generating the summary CSV-file. We worked around them by running the benchmarker software in *verbose* mode, where it generates a detailed log file. This raw log file contains all data before aggregation and therefore before any of the previous issues occur. Storing this log data allows us to run the aggregations ourselves and provides us with all available information, even when the software crashes. Additional summarizations are possible, since the log file is in fact much richer in information than the original CSV-file.

The post-processing pipeline parses this log file and converts it into a more detailed CSV file, which contains *query events*. These events contain the essential information of a single query execution. Query events serve as the basis for all results in this research paper. The schema of a single query event is shown in Table 7.9. All event files and derived views are online [26].

Table 7.9: Content of the query events used as the starting point for all benchmark results in this work

Field	Range
sim_id	(engine, number of nodes, memory, config.)
query_name	400 ids for WatDiv, 1,223 for Ontoforce
thread_id	6 ids
thread_type	warm-up (1 thread) or stress (5 threads)
order_id	the offset in the querymix for a thread
number_of_results	-1 if error, ≥ 0 otherwise
runtime	(seconds), error: -1, timeout: max. value
flag	SUCCESS, ERROR, TIMEOUT
correct	(IN)CORRECT (if #results $\neq$ consensus)

The query events can also be used to study query correctness, since they contain the number of results per query and a flag for (un)successful query execution. For the Ontoforce benchmark however, almost half of the queries are COUNT queries, for which the result count does not provide any guarantees on correctness. To verify

[26] http://users.elis.ugent.be/~drdwitte/resultscsv.html

the correctness of these queries, we extended the benchmarker software enabling it to store the actual query results, which allows us to compare the results of the COUNT queries.

To simplify the deployment of this modified benchmarker client, we automated this process by creating a Docker container which automatically installs the software and its dependencies.

Finally, we automated major parts of the post-processing of benchmark results, because (i) this saves the future benchmark user a lot of time parsing the benchmarker log files; (ii) provides the user with a large set of instant visual results; and (iii) allows knowledge-transfer to new benchmarking efforts through script re-use.

Jupyter notebooks[27] were used for the post-processing All notebooks are available online [28].

Practice has shown that the event format leaves room for unanticipated analysis. For example: dealing with incorrect queries, taking into account server load or caching effects, studying the reason behind one of the query engines crashing,...

7.5 Results I: Approaches to Linked Data at Scale

In this section we will study the query runtime distributions of different approaches for dealing with Linked Data at scale. In the figures both the *median* and *mean query run-times* are reported. As the runtime distributions can be skewed, performance differences between systems are most often reported using the median runtime.

If we consider an ETL[29] process, or equivalently a batch of queries, the mean runtime is more meaningful, as it directly translates to the total batch runtime. In the following box plots we chose to report both.

Some of the stores provide query results in a streaming fashion. Response times are not captured in the current query event format but are captured in the SPARQL benchmarker summary CSV-files. For GraphDB and Blazegraph the response times are respectively 27% and 21% lower than the mean run-times on WatDiv1000M. For the other engines the difference was close to zero.

A major concern when comparing query runtime between different engines is *query completeness*. The current query event format, shown in Table 7.9, explicitly reports

[27] `http://jupyter.org/`
[28] `http://users.elis.ugent.be/~drdwitte/postprocessing.html`
[29] Extract-Transform-Load: `https://en.wikipedia.org/wiki/Extract,_transform,_load`

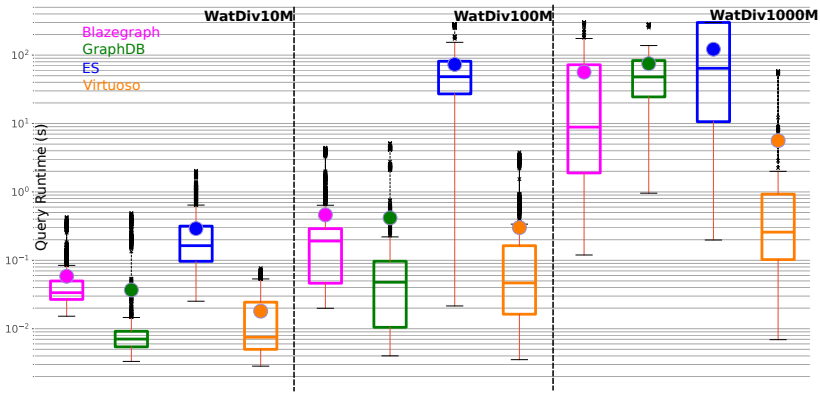


Figure 7.12: Query runtime distributions of *Vendor* systems for 3 different sizes of WatDiv. Dots correspond to average run-times, while the horizontal lines in the box plots correspond to median run-times The difference is scaling behavior between **Vir_32** (linear) and the other stores emphasizes the different impact of server memory on runtime behavior. **Bla1_32** and **Gra1_32** are very close for batch workloads, for individual queries GraphDB is superior except when scaling up to WatDiv1000M. **ES1_32** is the only store with timeout problems starting from WatDiv100M.

whether a query was solved correctly, meaning it has retrieved the complete set of results. Query completeness is the topic of section 7.6. To interpret the results in this section correctly, it is important to understand that queries, which have incomplete results for at least one benchmark, are completely discarded in the runtime comparisons.

Table 7.10: Conventions for describing benchmark setups. A description consists of a 3-character prefix describing the RDF storage solution, the number of nodes, the amount of memory and the configuration.

Shorthand Notation	**Full Description**
Vir1_32_Def	Virtuoso - single node - 32GB RAM - Default Configuration
TPF3_64_Def	Triple Pattern Fragments - 3 slave nodes - 64GB RAM - Default Configuration
Gra1_64_Opt	GraphDB - single node - 64GB RAM - Optimized (RFI) Configuration

Finally, a subtle error can be made in query runtime comparison for benchmarks

which involve a query engine that becomes unresponsive (engine failure). In the runtime comparisons we only consider the range between the first query and the last successful query. We coin this the *benchmark survival interval*.

In Table 7.10 we introduce a naming convention to describe the different benchmark setups.

Increasingly Large Datasets

The previous benchmark results [14] stem from the *None* configuration. In this section however, we use the *Default* configuration of the *Vendor* systems. In Figure 7.12 query runtime distributions are shown of the 4 *Vendor* systems for three different dataset sizes of the WatDiv benchmark: 10M, 100M, and 1000M (million) triples.

- **Runtime vs Dataset Size:** Although only 3 data points are available for 10, 100 and, 1000M triples, it is interesting to investigate how the runtime scales when the dataset grows by a factor 10. If we focus on the average query run-times (dots) two trends can be observed: **Vir1_32** has a nearly constant multiplication factor (mf) while for the other stores this is not the case. Going from 10M to 100M the mfs are 8, 11, and 17 for **Bla1_32**, **Gra1_32**, and **Vir1_32** respectively. Going from 100M the mf for **Vir1_32** is 19, but for the other systems mf $>$ 120! A possible explanation for this trend break is that memory swapping occurs. This observation motivates the choice for 64GB memory instances as the central reference setup from which to explore the benchmark space.
- **Timeouts & Errors:** Most of the queries are executed successfully by all *Vendor* systems. For WatDiv1000M **Bla1_32** already has a timeout percentage of 11.6% and for **ES1_32** this is even 32.7%. Note however that these results do not affect the plots as we only use query events from the *benchmark survival interval*.
- **GraphDB vs Virtuoso:** In terms of median runtime both **Gra1_32** and **Vir1_32** are tied at 0.01s and 0.05s in the two leftmost panels of Figure 7.12. With sufficient memory these engines can remain competitive. However, in the 32GB setting only **Vir1_32** is performant, with a median runtime of 18.6s. **Gra_32**, more than the other stores, suffers from a slow tail which has a major effect on the mix runtimes. There Virtuoso is 1.5-2 times faster.

- **Blazegraph vs. GraphDB: Bla1_32** competes with **Gra1_32** for batch workloads but not in terms of median runtimes.

- **ES consistently last: ES_32**, even on WatDiv10M, lags by a factor of at least 5 to the other systems. For WatDiv100M already the nonlinear scaling behavior sets in, making it the only engine to experience problems with the 100M dataset.

Vertical Scaling

In the previous section we saw that the amount of memory is a critical parameter for benchmark performance. In this section we study the effect on the query runtimes of increasing the amount of memory to 64GB. The two leftmost panels of Figure 7.13 study the effect of vertical scaling.

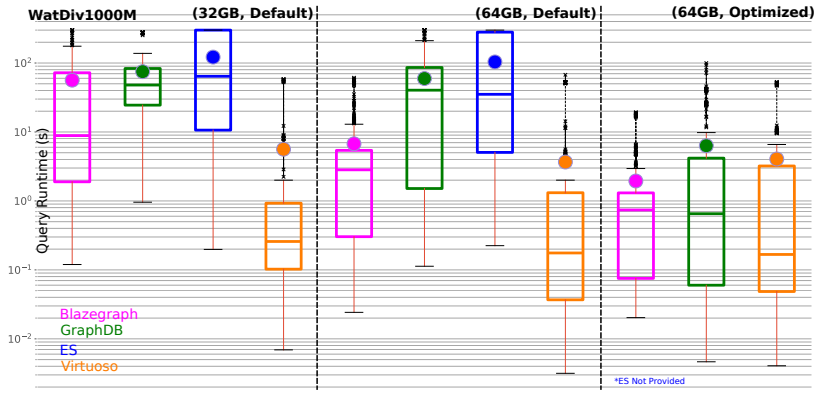


Figure 7.13: Query runtime distributions for WatDiv1000M showing the effect of increasing memory from 32GB (left) to 64GB (center) and *Optimized* configurations (right). Virtuoso hardly doesn't benefit from additional memory or better configurations. GraphDB is the most sensitive to proper configuration. In the right panel engine performance starts converging. For batch workloads **Bla1_64_Opt** is the fastest, in terms of median runtimes both **Vir1_64_*** setups perform best.

- **Memory is no magic solution:** Especially for **Gra1_64_Def** and **Vir1_64_Def** hardly any improvement can be seen. Blazegraph takes full advantage of the additional memory, with a large shift in both median and mean runtimes. The strong hardware dependence of Blazegraph could be a motivation to also

study the performance in a GPU setting[30], which is outside the scope of this work.

- **Speedups: Bla1_64_Def** has a speedup of 8.4 for its average runtime and 3.1 for its median runtime. From the other stores only **ES1_64_Def** benefits with a speedup of 1.8 for its average runtime.

- **Benefits for fast queries:** The most outspoken positive effect is the lowering of the lower boundary of the box plots.

RFI: Optimized Configurations

After contacting the vendors with our initial results one of the parties suggested to demonstrate the optimal operation of their database. This was formalized by sending out a Request-For-Information (RFI), specifying the benchmarks we were planning to run. 3 out of 4 vendors chose to participate in the RFI, which resulted in an *Optimized* configuration.

In Figure 7.13 the rightmost panel corresponds to running the benchmark with the *Optimized* configuration.

- **Sensitivity to configuration: Vir1_64** got no benefit from the RFI settings file. For **Bla1_64** the only improvement was to explicitly configure the time-out parameter on the server side. This avoids unnecessary overhead while the client was already disconnected and leads to a speedup of approximately 3.5 for both runtime measurements. **Gra1_64** has the highest sensitivity to proper configurations. The provided scripts ensure a batch speedup of 9.4 and a median runtime speedup of 62.

- **32_Def to 64_Opt:** Moving from the left panel to the right in Figure 7.13, we clearly see results converging. **Bla1_64** is the most efficient system for processing batch workloads with an average runtime of 1.95s per query, 4.05s and 6.32s for **Vir1_64** and **Gra1_64** respectively. In the query performance **Vir1_64** has a median runtime of 0.17s where **Gra1_64** and **Bla1_64** have runtimes of 0.65s and 0.74s respectively.

[30] `https://www.blazegraph.com/whitepapers/Blazegraph-gpu_InDetail_BloorResearch.pdf`

- **Runtime vs Dataset Size:** Returning to section 7.5 we can verify that the linear scaling behavior is largely restored, confirming our earlier hypothesis. Multiplication factors drop to 4.2 for Blazegraph, for Virtuoso and GraphDB mf $\approx$ 15.

- **Timeouts & Errors:** Apart from 5% timeouts for **Gra1_64_Opt**, no query errors are observed with the *Optimized* configurations.

Semantic Web Solutions

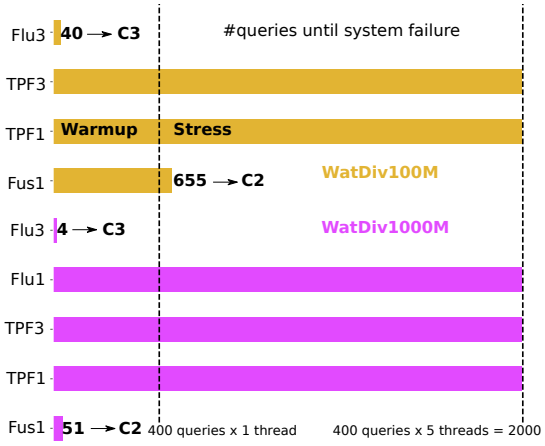


Figure 7.14: Benchmark survival interval for 3 *SemWeb* solutions. For early crashes the amount of queries until system failure is reported, as well as the query template causing the failure. **Flu3_64** crashes upon the first occurrence of a **C3** query. **Fus1_64** survives the warm-up run for WatDiv100M but crashes upon the first occurrence of a **C2** query in the stress test, for WatDiv1000M again the first **C2** query in the warm-up run causes the crash.

As the initial goal of the research collaboration with Ontoforce was to find a solution to work with federated querying on top of live data sources on the Semantic Web, we discuss the results of **Fus1_64**, **Flu3_64**, **TPF1_64** and **TPF3_64**. Figure 7.14 deliberately has no relation with query runtimes. For these 3 systems engine failure and query errors are very common with only the **TPF*_64** systems surviving the entire benchmark.

- **Flu1_64:** The federation system with 1 source node is added to verify that FedX in fact manages to parse the queries.

- **Specific Templates cause crashes:** Where **TPF*_64** systems more gracefully timeout on the **C** templates, **C2** causes a crash in **Fus1_64** and **C3** in **Flu3_64**, upon their first occurrence in warm-up or stress run. **C3** is a query with very low triple pattern selectivity leading to large in-memory joins.

- **Crash investigation:** For **Flu3_64** the benchmark was terminated after running into constant timeouts for 8 hours. Upon inspection of the slave nodes (VOS), these turned out to be idle, while the federator node had its entire memory pool saturated, with the CPU load close to zero. A possible explanation might therefore point in the direction of issues with garbage collection. For **Fus1_64** after a number of queries a continuous timeout sequence sets in. Peculiar was that the specific HDT implementation for Fuseki seemed to ignore the timeout parameter which might explain why the server overloaded and became unresponsive.

- **Staying alive: TPF*_64** survive both WatDiv benchmarks, nonetheless up to 71% of the queries timeout for WatDiv1000M. For WatDiv100M the timeout ratio drops to 25% for **TPF3_64** and to 11% for **TPF1_64**.

- **Runtime Comparison:** Only for WatDiv100M comparing the runtimes of **TPF*_64** to the *Vendor* systems is meaningful due to the higher query success rates. Compared to **ES1_32**, the **TPF1_64** is 2.4 times faster in terms of median runtime and 12% in terms of batch time. For **TPF3_64** the results are worse than **ES1_32**: 25% slower in median runtime, 40% slower for batch.

Horizontal scaling

An alternative to increasing the memory in a single-node server, is to increase the overall resources, by adding more nodes, thus creating a distributed system.

All 4 commercial RDF solutions support multi-node setups. GraphDB however, works only as a HA-solution (High-Availability): We did not evaluate this approach since it requires all data to be replicated on every node and does not support data partitioning, which is required to scale beyond the single-node resource limits. The performance can however be estimated since it is equivalent to a setup with N identical databases, with a load balancer equally distributing the queries between the database replicas. The effect is a linear speedup, in terms of completing a full

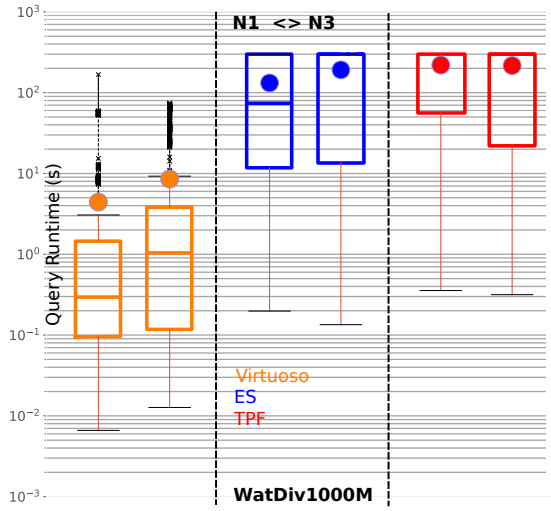


Figure 7.15: Pairwise comparison of query runtime distributions for single-node versus 3-node setups. None of the solutions achieve an average runtime speedup when adding more nodes, on the contrary overhead multiplication factors of 1.9 and 1.5 are seen in left and center pane for **Vir3_32_Def** and **ES3_32_Def**. For **TPF3_64_Def** the overhead is negligible.

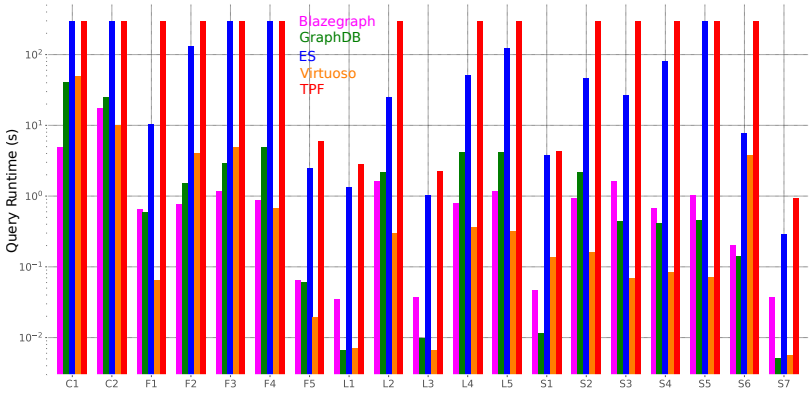


Figure 7.16: Average Runtime per query template for 5 single-node setups. **TPF1_64** has only 5 templates which do not coincide with the timeout of 300s, for **ES1_64_Def** this is already 15 templates. **Vir1_64_Opt** is the fastest engine for 13 templates, **Gra1_64_Opt** for and **Bla1_64_Opt** for 3 templates each. Template **C3** was omitted due to query completeness issues. Blazegraph was the only engine to retrieve all results.

querymix. Virtuoso also supports a similar setup. The effect on the individual query runtimes should be limited, but not completely absent since the database load on the individual nodes will be smaller. The effect of database load on the query runtimes will be studied in the next section.

For Blazegraph support is required for setting up the multi-node system. This support was requested via the RFI but not fulfilled, which limited our comparison to **Vir3_32_Def**, **ES3_32_Def**, and **TPF3_64_Def**.

In Figure 7.15 we show pairwise comparisons of the three setups for which we have both a single and a 3-node benchmark.

- **Benchmark survival interval: Vir3_32_Def** and **TPF3_64_Def** managed to stay online during the entire Watdiv1000M benchmark, **ES3_32_Def** stopped responding after having completed 67% of the multi-threaded run.
- **Errors & Timeouts:** 65% of the queries of **ES3_32_Def** resulted in an HTTP 504 error, mentioning *Gateway Timeout*. Further study revealed that this timeout was due to an internal configuration parameter in the ES distributed setup, unfortunately we did not receive any feedback on this issue. **Vir3_32** successfully completed all queries. 70.6% of the queries result in a timeout for **TPF3_64_Def**.
- **Multi-node overhead:** For all setups additional nodes lead to overhead instead of runtime speedup. Runtime multiplication factors are 1.9 and 1.5 for **Vir** and **ES**. **TPF** has a negligible overhead but is already very close to the query timeout.

In a discussion with OpenLink it was clarified that Virtuoso Cluster acts as a *distributed memory solution*. This implies that adding nodes does not lead to a speedup in the query runtimes, but the total of memory pool in the system increases, allowing it to handle larger datasets for which a single node instance might not be suited. Since the single node benchmark did not exhaust the memory, there is no advantage to be expected from a multi-node setup. As an indication, according to support a 32GB machine should be able to manage up to 3 billion triples (10GB per 1B triples). This observation, together with the lack of feedback on the issues with **ES3_32_Def** and the high timeout percentage for **TPF3_64_Def** motivated our decision to not run any additional benchmarks with this approach for WatDiv1000M.

Systems translating SPARQL queries to distributed platforms such as Hadoop [12, 22] are an alternative approach we did not test. An example of such an approach is S2RDF [46]. For this solution, results on Watdiv1000M indicate that a 10-node setup can be close to 10 times faster than a single-node Virtuoso server. Since these SPARQL-on-Hadoop solutions are not sufficiently mature and for example cannot be tested using a SPARQL endpoint definite conclusions can currently not be drawn. One observation to motivate this caution is the fact that Virtuoso is hardly affected when running multiple benchmark clients at once, as will be shown in section 7.6. The operational cost for these Hadoop setups can also not immediately be deduced.

7.6 Results II: Query Runtime Analysis in Depth

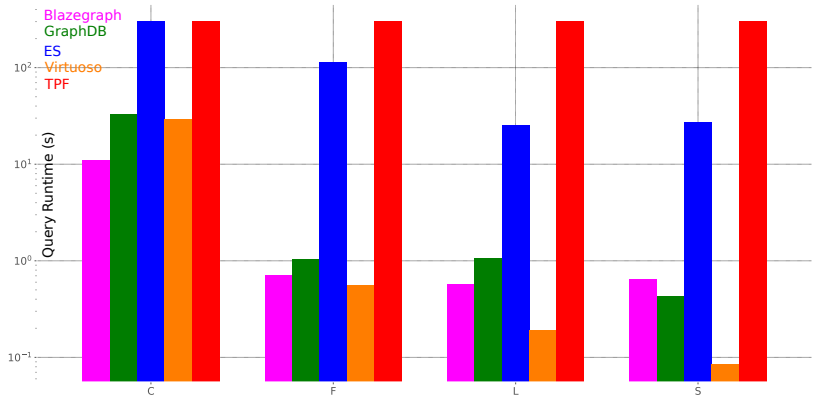


Figure 7.17: Average Runtime per BGP type.

Only comparing query runtimes might be an oversimplification in benchmark analysis. When comparing batch runtimes the slow tail of the distributions dominates the average runtimes. In section 7.6 we investigate whether certain query templates dominate the batch runtimes. Query execution times depend on the state of the database, which motivates studying the query context. Previous results are still valid as all queries are executed 5 times and each time the median is taken to calculate batch runtimes.

In section 7.6 we study the effect of the server load on the query runtimes by comparing a single-client benchmark with a stress test with 5 clients. What queries have been executed previously might also affect query runtimes due to result caching. This will be explored in section 7.6. Often not reported, but the effect of result

completeness can have a big impact on the query runtimes reported, as we will discuss in section 7.6.

Query runtimes for different template types

The queries of the WatDiv benchmarks are all BGPs but have different shapes and selectivity properties. The benchmark generator has 20 templates which can be further organized into 4 template categories (shapes). In Figure 7.17 we show the average runtime per template for 5 stores on WatDiv1000M.

- **Template timeouts:** For **TPF1_64** 15 runtime averages coincide with the benchmark timeout (300s). Successful queries are spread out over the different types: **F**:1, **S**:2, **L**:2. **ES1_64_Def** has timeouts for the 2 **C** queries, 2 **F** queries, and 1 **S** query. The other stores have no averages close to timeout.
- **Template winners: Vir1_64_Opt** is the fastest engine for 13 templates, nonetheless **Bla1_64_Opt** was better on batch workloads. These workloads are dominated by the runtimes of the **C**-templates, more specifically **C1** seems to explain the difference. **Gra1_64_Opt** performs best for 3 **S**-templates, **Bla1_64_Opt** wins on 1 **C**- and 2 **F**-templates. Template **C3** was omitted due to query completeness issues. Blazegraph was the only engine to retrieve all results within the timeout boundary. **Vir1_64_Opt** wins: **C**:1, **F**:3, **S**:4, and all **L**-templates.

If we generalize further and only distinguish between 4 query template types, as can be seen in Figure 7.17, it becomes even more apparent where the difference between Blazegraph and Virtuoso can be situated: the **C**-templates.

- **Ranking per template type:** This pattern is very stable, **Vir1_64_Opt** first, followed by **Bla1_64_Opt** and **Gra1_64_Opt**. Only for **C**-templates Blazegraph has the advantage by a factor 3: 10s vs 30s. The differences on the other templates are lower by an order of magnitude, each time in the range of 0.2 - 0.5 seconds. For the **S**-templates GraphDB performs slightly better than Blazegraph.
- **Engine specialties:** For Blazegraph the **C**, **F**, and **S**-templates result in similar runtimes. GraphDB has a small preference to **S**-templates. Virtuoso is much

better than the competition for **L**-queries and **S**. For the **F**-template all three engines perform similarly.

Single versus Multi-client stress testing

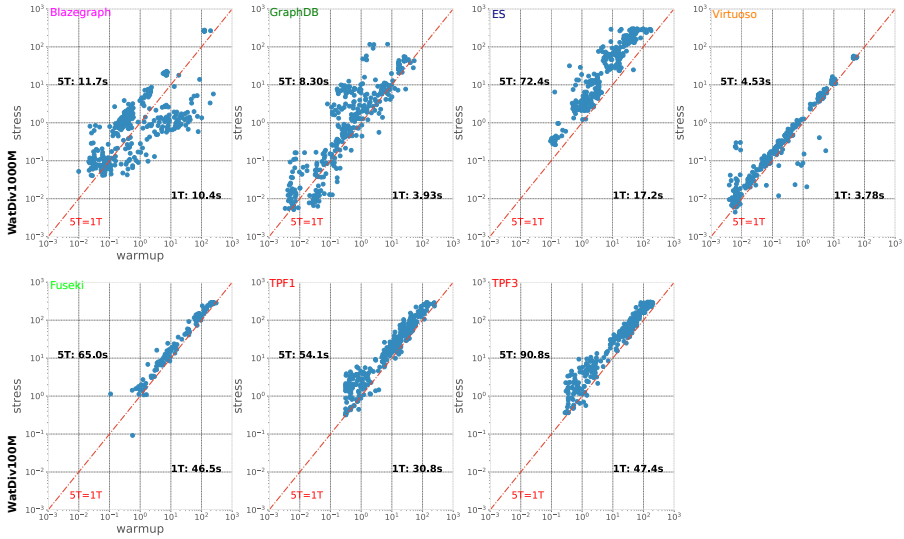


Figure 7.18: Runtimes for single versus multi-client workloads: 1 vs. 5 threads. 5T runtime corresponds to the maximum runtime per query in the stress test, 1T is the runtime during the warm-up phase. The red line corresponds to the bisector, where the runtime for both workloads is equal. Dots are expected to be shifted up, which correspond to a multiplication factor. The closer the dots to the bisector the smaller the multi-client overhead. Dots below the bisector can be attributed to the natural variance in query runtimes. Average runtimes per store are also shown. **Bla1_64** and **Vir1_64** have the smallest overhead ($< 20\%$), for **ES1_64** has the largest ($> 300\%$).

All results so far focused on the multi-threaded benchmark run, in which 5 benchmark clients are simultaneously executing the same querymix in a (different) randomized order. It is however interesting to take into account the effect of server load. In Figure 7.18 we compare, per query, the runtime of the warm-up run versus the runtime of the slowest multi-threaded run. We chose the slowest query as this has the highest probability of eliminating the effects of caching which will be studied in the next section. Note that for the *SemWeb* systems the comparison is on WatDiv100M, while the *Vendor* systems are compared on WatDiv1000M.

- **Highest resilience against server loads**: The lowest multiplication factors

(mf) are 1.1 , 1.2, and 1.4 for **Vir1_64_Opt**, **Bla1_64_Opt**, and **Fus1_64_Def** respectively.

- **Lowest resilience against server loads:** For **TPF*_64_Def** the mf is 1.8 - 1.9. **Gra1_64_Opt**'s mf is at 2.1, but for **ES1_64_Opt** we have an mf of 4.2.
- **Variance of query runtimes:** For Blazegraph and GraphDB the variance on the query runtimes might still be explained as the result of caching. As we will see in the next section however, caching only plays a role for the slow-running queries (**C**-templates) in the case of Blazegraph.

The Role of Caching in Query Runtime Results

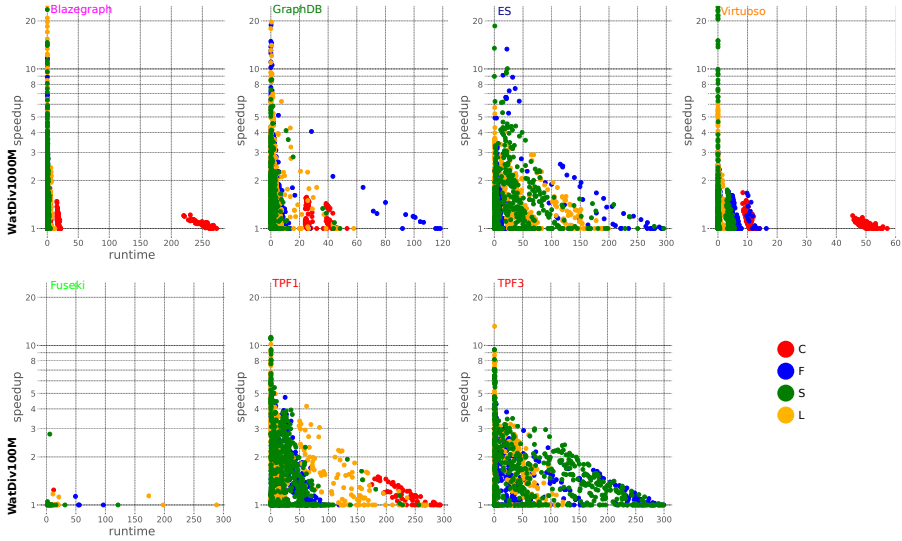


Figure 7.19: Speedup in query runtime by comparing all query runtimes in the multi-threaded run with the slowest execution in the stress test. With no caching all dots are expected on the X and Y-axis, the latter because of the noise on small query runtimes. If we focus on speedups > 2, especially **ES1** and **TPF*** seem to have the highest benefit.

Some data stores cache the results of queries. Especially in a benchmark where the same query is executed multiple times, this might lead to a large variance on the query runtimes. Although the approach with query events was not designed with support for studying caching effects in mind, having the order of the queries suffices. In an initial attempt we plotted the query runtimes as a function of the

distance to their nearest preceding execution. For this distance we experimented with the number of intermediary queries, the total number of intermediary results, and the amount of time in between. Results were very similar but did not show any clear pattern. In Figure 7.19 however, the speedup with respect to the slowest query execution in the multi-client run is plotted as a function of the actual query runtime. This visualization allows an easy distinction between speedups which are caused by noise, mainly for very short query runtimes, and real caching effects. If no caching effects are present the plot should have all its dots on the X and Y-axis.

- **Stores with clear caching advantage:** The **TPF** server instances have NGINX[31] cache enabled. The similarity in results with other stores strengthens the idea that Figure 7.19 in fact shows caching behavior for **TPF*_64**, **ES1_64_Def**, and **Gra1_64_Opt**.
- **Caching differences per template type:** For **ES1_64_Def** and **Gra1_64_Opt** the **F**-templates (blue) dots correspond to the highest speedups. For **TPF*_64** query execution is in general slower than for the other systems, therefore **L** and **S**-queries, shift to the right and their speedups become more prominent. Small speedups for **Bla1_64** and **Vir1_64** are mostly limited to the **C**-template queries.
- **TPF1 vs TPF3:** As a result of the horizontal data partitioning scheme **S** and **F**-queries can be resolved locally for **TPF3_64** which explain the higher prevalence in the plot.

Query Result Completeness

In our SWAT4LS contribution [13] we discovered that query runtimes cannot be trusted without paying careful attention to query completeness. We revisited earlier results on WatDiv and discovered some inconsistencies as well.

- **Vir*_Def:** Running Virtuoso with the *Default* configurations gave it an advantage since in this setting the result count is limited to $100,000$. This only affects the **C3** queries for all sizes of WatDiv.

[31] https://www.nginx.com/

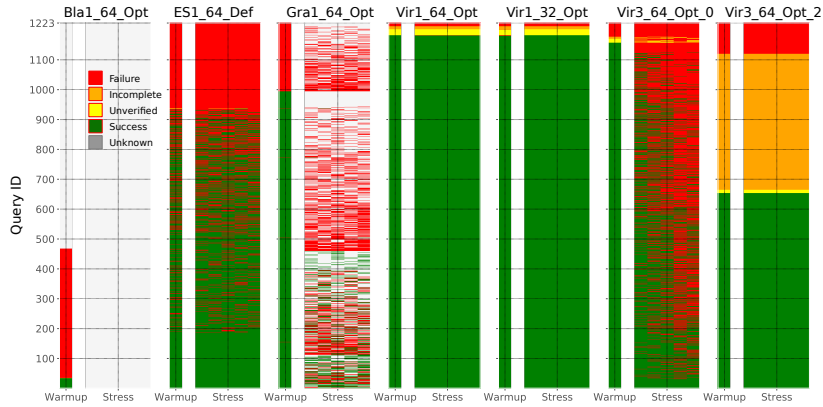


Figure 7.20: Overview of successes and errors per query (Y-axis) and thread (X-axis) on the Ontoforce benchmark. Queries are sorted per system in order to group error behavior and are not consistent between simulations! Blazegraph has a short benchmark survival interval. **ES1**, **Gra1** and **Vir3** Cluster setups have a lot of errors but most queries execute successfully at least once, which allows runtime comparisons. **Vir3_64_Opt_0** is the most successful Virtuoso cluster run as query completeness analysis revealed that **Vir3_64_Opt_2** has unreported errors for 37% of the queries.

- **Vir*_Opt: Bla1_64_Opt** was the only engine to correctly solve all **C3** templates. This query returns the highest number of results: 42,063,279. Although Virtuoso was configured to report an unlimited number of results, we discovered that for multiple independent queries the result count is limited to the magic number: 1,048,576. (which is 2^{20}).

As mentioned in the introduction of section 7.5, none of the runtime results reported so far are affected by this query incompleteness as we discarded all queries for which at least one store had a different number of results as compared to the consensus. In practice this means that all **C3** queries had to be discarded. The impact on the runtime comparisons is big as **C3** has the highest runtimes and ignoring query completeness would put Blazegraph at a serious disadvantage.

7.7 Results III: Real-world Life Sciences Benchmark Results

As was explained in section 7.4, the WatDiv benchmark and the Ontoforce benchmark are related. The Ontoforce benchmark consists of interactive federated queries which are extracted from the user logs of the DISQOVER product. These queries

are currently solved by combining an ETL preprocessing step, which integrates the different Life Sciences datasets offline using Ontoforce's own central ontology. This ETL step bares a lot of similarity with the WatDiv benchmark as it consists of mostly BGP queries. The queries of the Ontoforce benchmark are the result of faceted browsing, whereby, in practice, the facet filters are performed by a distributed search system (SOLR), but their product can also run with a SPARQL-based back-end. In this section we evaluate the ability of *Vendor* systems to work with these types of queries and therefore serve as an alternative to a search system.

The Ontoforce benchmark has a very challenging query set. Therefore the focus of the next section will be far less on query runtimes, but more on trying to extract insights which are generalizable. In this benchmark run, the response times consistently coincide with the query runtimes. In section 7.7, we give more detailed insights in the behavior of the different systems on the Ontoforce benchmark. We pay special attention to query failures and query completeness. In section 7.7, we try to automatically infer the reasons behind query success, failure, different error types, and slow versus fast running queries. This automation is achieved by making use of *decision tree analysis*. Finally, in section 7.7 we compare the results of all benchmarks in this research using *Benchmark Cost* as a unification parameter. This allows to make comparisons between setups which are very different in nature and see whether the trends in the benchmark results are consistent. This approach also takes into account the cost for data ingestion and the different licensing fees.

Benchmark Error Analysis

Error Frequencies The *SemWeb* systems and **Vir*** have been tested on the Ontoforce benchmark for our SWAT4LS [13] contribution. Note that **TPF** systems do currently not support all SPARQL operators and could therefore not be run on this benchmark. Only Virtuoso simulations had a sufficiently wide benchmark survival interval to enable further analysis. In Figure 7.20 we show the results for the *Vendor* systems. Each simulation consists of a small bar, corresponding to the single-threaded warm-up run, and 5 concatenated bars corresponding to 5 threads in the stress test.

- **Bla1_64_Opt:** One major difference with the results on the WatDiv benchmark is Blazegraph's inability to handle the complexity of the Ontoforce queries, resulting in very short benchmark survival interval: it contains only 55 queries, of which 18 are timeouts.

- **Gra1_64_Opt:** GraphDB also did not survive the entire benchmark, but managed to stay up for 21% of the stress run. During the stress run it solved 40% of the queries successfully, the other queries resulted in a timeout. For 38% of the queries, at least one successful run is available in the stress run.
- **ES1_64_Def:** ES was definitely the least successful on the WatDiv benchmarks, but is the only store, apart from Virtuoso, for which the benchmark survival interval spans the entire benchmark. 58% of the queries were executed successfully. The remainder consists of 25% HTTP errors and 17% timeouts.
- **Vir1_*_Opt:** Virtuoso is both consistent and successful on this benchmark with only 1% of queries consistently failing, overall the success rate is 98%. These failures correspond mainly to queries which contain *property paths*. None of the other stores could handle these queries. It should be noted that during the creation of the DISQOVER product, Virtuoso was frequently used as a back-end system, which partially implies a certain favorable bias in the Ontoforce results.The **Vir1_32_Opt** in the SWAT4LS [13] paper had 41% incomplete queries. This re-run however, achieves the same figures as the 64GB run.
- **Vir3_64_Opt_*:** Although the success rate of **Vir3_64_Opt_0** is only 55%, 92% of the queries are successfully executed at least once, which makes it possible to make runtime comparisons. **Vir3_64_Opt_2** has far less reported errors. Post-processing revealed issues with query completeness (orange) for 37% of the queries.

Query Correctness. Previously published results [13] had counter-intuitive runtime results: **Vir1_32** and **Vir3_64_Opt_2** executed much faster than **Vir1_64**. Consequently, we studied the number of results per query:

- **Inter-thread consistency:** As a first step we analyzed whether for each individual system the number of query results was consistent for each querymix. Without any exception this inter-thread consistency was confirmed.
- **Query consensus:** In the query event format, described in Table 7.9, one field indicates whether a query is correct or its result count incomplete. These values are obtained by creating a query consensus, with the following rules. If at least two separate *Vendor* systems agreed on the number of results we

assume this results is 'correct', for 97.3% this is the case. If only 1 engine can solve a query we label these as 'uncertain'. Virtuoso solves 19 queries for which no consensus can be derived. For 13 queries none of the systems managed to generate a solution. 8 of these contain a property path operator, the other 5 have FILTER IN operators containing large URI lists, such that the file size of the query is between 10 and 100 kb.

- **Count Queries:** Of the 19 'uncertain' queries solved by Virtuoso 15 are COUNT queries. However, upon inspection the COUNT operator was always part of a subquery, so this result can not be disproven. The benchmark software only reports the number of results per query. We extended it to also download the actual results to be able to verify whether the COUNT queries are consistent between the stores. However, no inconsistencies were found there.

- **Incorrect Queries:** Some of the Virtuoso benchmarks have incorrect queries. The typical pattern is that the query is executed $< 1s$ and generates 0 results. 1 query also had the query result limit $= 2^{20}$. To get more insight into the context of incomplete queries we executed the **Vir3_64** benchmark an additional 3 times. However, the cluster did not exhibit this behavior, but the new benchmarks never made it to the stress test, with the best run having a benchmark survival interval of length 228.

Decision Tree Analysis of Query Features

Ontoforce has released the queries for this benchmark run. However, the queries are very complex and sometimes they take up 1 - 100 kb in disk space. To gain a deeper understanding into why queries fail, have timeouts and HTTP errors, why they are fast or slow to execute,... we created a set of features per query and fitted a decision tree [32] to the data. The 3 resulting trees are shown in Figures 7.21 to 7.23 . The input features for this algorithm are limited by removing highly correlated features. For example ORDER and LIMIT are highly correlated. The list of retained query features is given in Table 7.11 together with the highest correlated operators. By adding 'Query Engine' as an additional feature we can train the decision tree on all the available query event data for the Ontoforce Benchmark. We also ran this analysis without this categorical features but the resulting rules were not informative.

[32] `http://scikit-learn.org/stable/modules/tree.html`

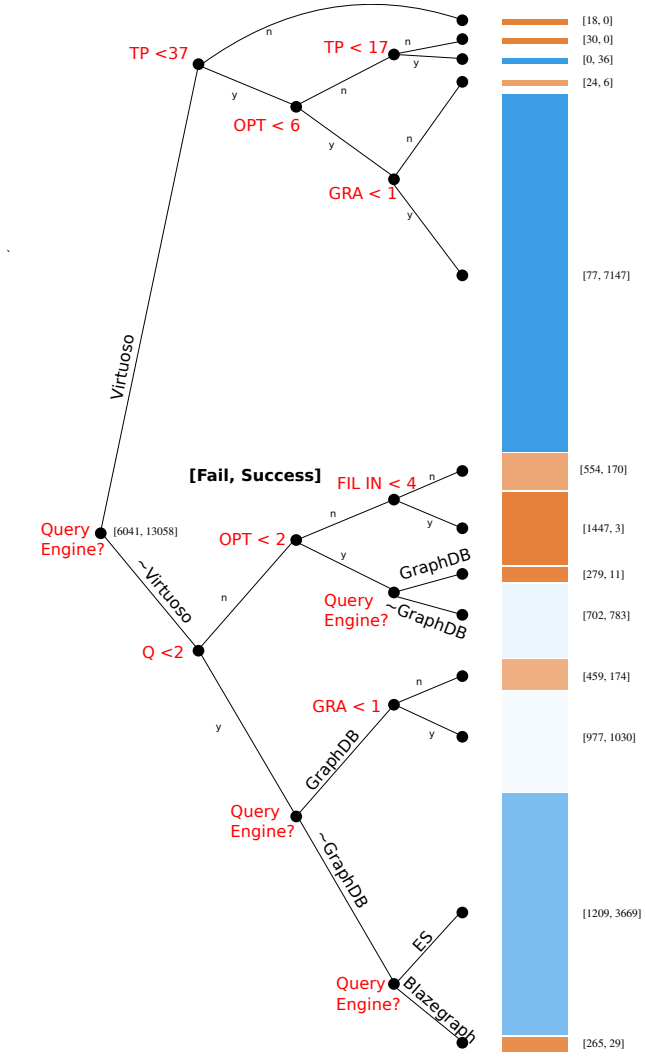


Figure 7.21: Decision Tree Analysis to identify the reason for query failure. Input for all trees are feature vectors, also the query engine is added as a categorical feature. Rules in the decision trees are shown in red, sample sizes are encoded as the height of the bar on the right-hand side. Classification into query success (blue) and failure (red) and incomplete. The query engine is an important decision rule, which demonstrates that Virtuoso behaves very different from the other systems.

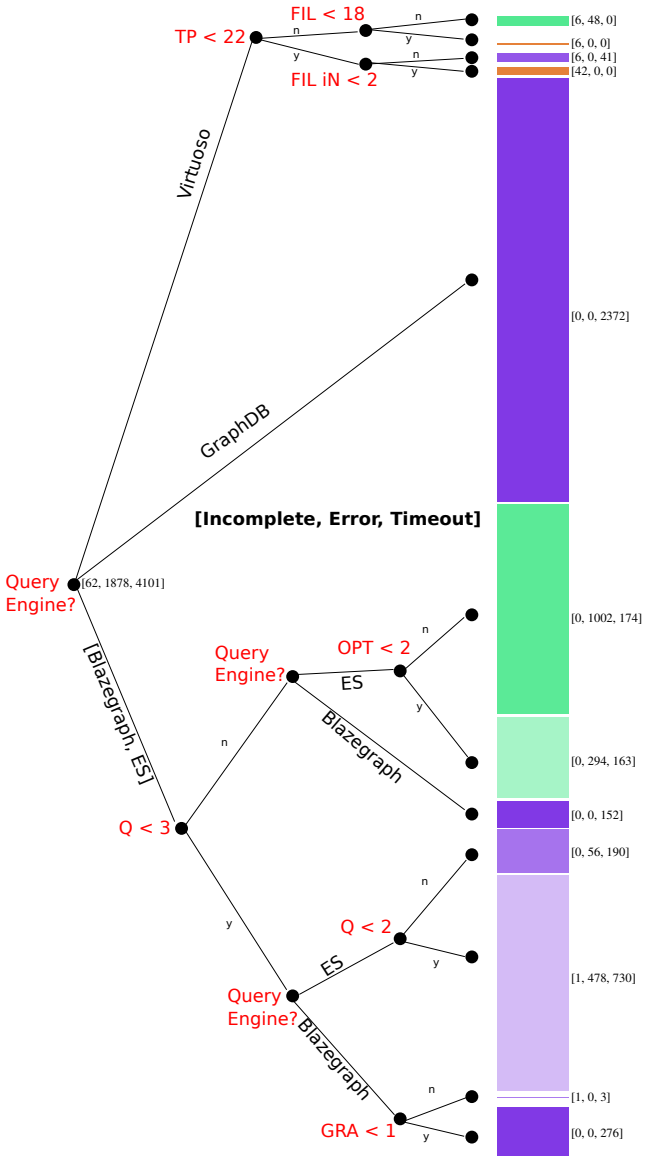


Figure 7.22: Decision Tree Analysis to identify the type of error which is most common if an engine fails. Input for all trees are feature vectors, also the query engine is added as a categorical feature. Rules in the decision trees are shown in red, sample sizes are encoded as the height of the bar on the right-hand side. Classification into query success (blue) and failure (red) and incomplete. The query engine is an important decision rule, which demonstrates that Virtuoso behaves very different from the other systems.

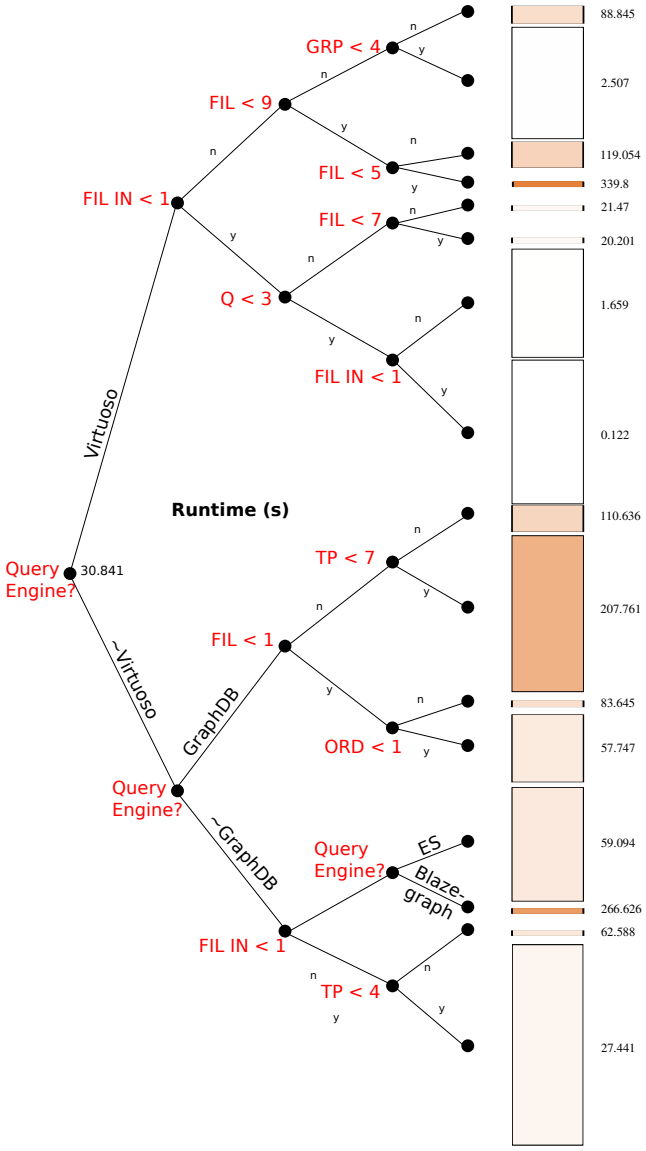


Figure 7.23: Decision Tree Analysis to identify the runtime of a certain query type. Input for all trees are feature vectors, also the query engine is added as a categorical feature. Rules in the decision trees are shown in red, sample sizes are encoded as the height of the bar on the right-hand side. Regression on query runtimes. Red corresponds to high query runtimes, white to low.

Table 7.11: Query Features and information on their range and correlations with other (discarded) features.

Feature	Prefix	Value	Range	Correlations
ORDER	ORD	frequency	[0,1]	LIMIT(0.88)
FILTER IN	FIL_IN	frequency	[0,16]	UNION(0.95), FS(0.95)
FILTER	FIL	frequency	[0,27]	tp_???(0.96), TP(0.95)
COUNT	CNT	frequency	[0,1]	DISTINCT(1.0)
Triple Patterns	TP	frequency	[1,38]	FILTER(0.95)
GRAPH	GRA	frequency	[0,1]	-
OPTIONAL	ORD	frequency	[0,9]	-
GROUP	GRP	frequency	[0,4]	-
(sub)Queries	Q	frequency	[1,10]	UNION(0.94), FILTER IN(0.94)
file size	FS	kilobyte	1, 10, 100	FILTER IN(0.97), UNION(0.95)
query engine	-	*Vendor*	-	-

- **Dominant Feature:** The 'Query Engine' is the most important factor to segment the data in all 3 cases. The absence of this feature would in fact indicate that all systems have similar behavior. **Vir1** thus is very different: it has fewer errors and query runtimes are significantly smaller.

- **Feature Importance:** If we take the number of node occurrences as a feature as a measure for feature importance then we see 3 features which occur in 5 nodes: TP, FILTER IN, FILTER. The FILTERS mainly play a role in the decision tree for runtime regression. In predicting failures and error types OPTIONAL, GRAPH and Q have the highest occurrences.

- **Highest failure rates:** The paths leading to samples with a high failure rate generally contain OPTIONAL operators. All engines except for **Vir1** suffer when $Q > 1$. **Gra1** also has a high failure rate for COUNT queries.

- **Most frequent error types:** For **Bla1** and **Gra1** the errors are all timeouts (purple). For **ES1** having multiple subqueries often leads to HTTP errors (green).

- **Queries with high runtimes:** For **Vir1** and **ES1** the FILTER IN operators are the main cause for high runtimes. For **Gra1** the presence of FILTERS pushes runtimes above 100s.

Finally we also investigate if the incorrect queries in the **Vir3** benchmarks had specific query features. Curiously, the problematic queries correspond to the most simple queries: $TP \leq 2$.

Benchmark Cost

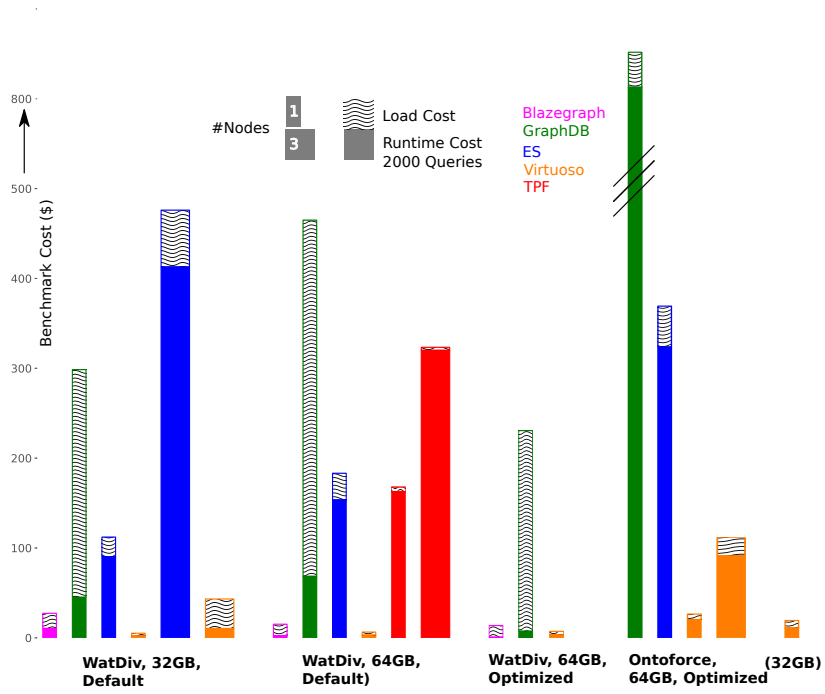


Figure 7.24: Benchmark Cost in $ to load and execute 2000 queries in a stress test for WatDiv1000M or Ontoforce datasets for different setups. All stacked bars consists the load cost stacked on top of the runtime cost. Bar width encodes the amount of nodes. For WatDiv **Vir1_32_Def** is the least expensive solution, mainly because **Bla1_64_Opt** has a much higher load cost. Also for the Ontoforce benchmark **Vir1_32_Opt** is the most cost-effective choice. The engine ranking is not conserved going from artificial to real-world benchmark.

In this section we aim to get a satellite view on the entire set of benchmarks conducted within this research. The penultimate trade-off for many applications in production is the cost for processing a certain workload. Our choice for using cloud hardware and AMIs enables this integrated view on all benchmarks: using cost we can compare single to multi-node setups, the cost for vertical scaling,...

All costs per store and for all benchmarks are shown in Figure 7.24. Costs stem from

an hourly price for servers on Amazon EC2, together with an hourly license cost for the AMIs.

The instance cost of the AWS hardware was $0.333 /hr for the 32GB server instances and $0.667 /hr for the 64GB instances. The licensing costs for the PAGO instances can be found on AWS marketplace and typically scale with the amount of memory per instance. For the 64GB instances, GraphDB's license cost is $1.4 /hr, for ES $2 /hr and for Virtuoso $0.80 /hr. Other systems tested have no licensing cost.

Additionally, before running a benchmark, the data has to be ingested in the system. This cost is stacked on top of the query cost in Figure 7.24. For some cases the ingest cost is unimportant as reloading the data is required only rarely.

- **The price of vertical scaling:** Is adding more memory, and therefore a higher license and infrastructure cost a wise choice? If we focus on the *Optimized* configurations for Watdiv1000M both **Bla1** and **Gra1** have lower operational costs when running the higher end hardware. For **Bla1** the price is lowered from $27 to $13.5, for **Gra1** the reduction is from $298 to $230. For the latter mainly the bulk loading process makes it less competitive. For **Vir1** the price goes from $5 to $7.

- **The price of horizontal scaling:** As adding more nodes led to higher runtimes, this also translates to higher costs. For **TPF** the costs go from $168 to $323 ($\times 1.9$) , for **ES** the costs rises from $112 to $475 ($\times 4.2$) and for **Vir** from $5 to $42 ($\times 8.4$)

- **The price for data ingestion: Gra1** seems to have the highest cost for loading the datasets, except for the Ontoforce benchmark. This is interesting as the Ontoforce benchmark has a much bigger dataset (2.4 BT). A possible explanation is that **Gra1** has trouble ingesting a single compressed (gzip) turtle file as was the case for WatDiv, while the Ontoforce dataset was ingested as 42 compressed (gzip) N-Quads files. For **Gra1_64_Def** many additional indexes are generated during ingest, which explains the lower cost for **Gra1_64_Opt**. Having more memory by itself can also impact the ingest process, for **Bla1** the ingest cost is lowered from $16 to $12. Virtuoso's bulk loader process is a real trump card in the cost comparisons. The load cost is $2.8 while **Bla1** in the optimal case has a cost of $12.6. The load cast is in fact larger than the runtime cost in this comparison. Also for the multi-node setups no advantage

is obtained in the ingest phase. **Vir3** takes 4 times more time to ingest while the cost/hr is also 3 times higher. For **ES3** a 33% cost increase is measured, while for **TPF3** the ingestion becomes 50% cheaper. The latter however is not **TPF** specific as the ingestion corresponds to the partitioning and compression of the data with the HDT algorithm.

- **The most cost-effective solution: Vir1_32** is the cheapest solution both for WatDiv1000M as for the Ontoforce benchmark, with costs of respectively $5 and $19.

7.8 Conclusion

In this chapter we offer guidelines and tools to run a *reproducible* benchmark. For the back-end we recommend working with hardware available via cloud providers, AMIs and Docker images for the system installation. We recommend releasing the configuration details for every store.

To enable critical reviewing benchmark output data should be released in its rawest form. The query event data in this work turned out to be an enabler for new unanticipated research questions. One example in this work is the study of caching effects.

The methods to arrive at certain research visualizations should be made available, which also provides knowledge transfer to future benchmark efforts.

In order to learn from challenging benchmarks, the benchmark approach should anticipate the occurrence of all sorts of errors. The information in these incomplete benchmark runs is in fact very valuable.

What are the trade-offs associated with certain setups? For every store we show the effect in terms of throughput and cost for vertical and horizontal scaling. Overall, the low-end setup **Vir1_32** gave the best results. For the other stores the best results are achieved with more memory and *Optimized* configurations. Benchmark cost allows the comparison of a heterogeneous mix of RDF storage approaches. *SemWeb* systems, of which **TPF*_64** performed best in this study, still lag by an order of magnitude in terms of performance with the *Vendor* systems. The research community would benefit from more realistic and challenging benchmarks, as it might stimulate the further development of current prototypes up to a level where they can compete with existing *Vendor* systems.

Future benchmarking efforts should consider, at least locally, scanning a benchmark space. This necessity was demonstrated in this work, by showing the effect of dataset size and by modifying the amount of server memory. An interesting result in this aspect is, that the performance of the different *Vendor* systems converged as they were given better hardware and configurations.

In answering the third research question, we demonstrated that query runtimes are an oversimplified representation of performance. Many contextual factors influence the runtime comparisons: certain query types might completely dominate the mix runtimes, server load and caching effects have a different impact of the systems tested. Adding query completeness analysis, makes benchmark runtime results more trustworthy. Ignoring this aspect, would have led to very different conclusions in this work.

The ranking of the different systems is not consistent if we change from artificial to real-world benchmarks. This supports the advice, to try and run use case specific benchmarks, before deciding on a system architecture. Although it was difficult to extract transferable insights from the Ontoforce benchmark, the decision tree approach, shed some light on certain SPARQL query features, which pose more problems to one system than another, giving the vendors some direction in optimizing their RDF storage solution.

As for the future work, the results in this work definitely indicate a lot of room for improvement in multi-node RDF storage solutions. While Virtuoso's offering is the most advanced, it is not yet as stable as its single-node counterpart.

References

[1] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. *International Semantic Web Conference*. Springer, pages 18–34, 2011.

[2] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of RDF data management systems. In: *The Semantic Web–ISWC 2014*, pages 197–212. Springer, 2014.

[3] R. Angles, P. Boncz, J. Larriba-Pey, I. Fundulaki, T. Neumann, O. Erling, P. Neubauer, et al. The linked data benchmark council: a graph and RDF industry benchmarking effort. *SIGMOD Rec.* 43(1):27–31, May 2014.

[4] E. Antezana, M. Egaña, W. Blondé, A. Illarramendi, I. Bilbao, B. De Baets, R. Stevens, V. Mironov, and M. Kuiper. The cell cycle ontology: an application ontology for the representation and integrated analysis of the cell cycle process. *Genome Biology*, 10(5):R58, 2009.

[5] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24, 2009.

[6] P. Boncz. LDBC: benchmarks for graph and RDF data management. *Proceedings of the 17th International Database Engineering & Applications Symposium*. IDEAS '13, pages 1–2, ACM, Barcelona, Spain, 2013.

[7] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: hyper-pipelining query execution. *CIDR*. Volume 5 of pages 225–237, 2005.

[8] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: a generic architecture for storing and querying rdf and rdf schema. *International semantic web conference*. Springer, pages 54–68, 2002.

[9] U. Consortium et al. Uniprot: a hub for protein information. *Nucleic acids research*, 2014.

[10] S. Coppens. Querying the linked data graph using owl: sameas provenance,

[11] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, Keppmann, et al. NoSQL databases for RDF: an empirical evaluation. In: *The Semantic Web–ISWC 2013*, pages 310–325. Springer, 2013.

[12] O. Curé, H. Naacke, M. A. Baazizi, and B. Amann. On the evaluation of RDF distribution algorithms implemented over apache spark, 2015.

[13] D. De Witte, L. De Vocht, K. Knecht, F. Pattyn, H. Constandt, E. Mannens, and R. Verborgh. Scaling out federated queries for life science data in production. *Proceedings of the 9th International Conference on Semantic Web Applications and Tools for Life Sciences*, December 2016.

[14] D. De Witte, L. De Vocht, R. Verborgh, K. Knecht, F. Pattyn, H. Constandt, E. Mannens, and R. Van de Walle. Big linked data etl benchmark on cloud commodity hardware. *Proceedings of the International Workshop on Semantic Big Data*, pages 6, Association for Computing Machinery (ACM), 2016.

[15] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 145–156, 2011.

[16] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz. The LDBC social network benchmark: interactive workload. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, pages 619–630, 2015.

[17] J. D. Fernández, M. A. MartıÍnez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 19:22–41, 2013.

[18] S. Ferré, A. Hermann, and M. Ducassé. Semantic Faceted Search: Safe and Expressive Navigation in RDF Graphs. Research Report PI 1964. January 2011, pages 27. `https://hal.inria.fr/inria-00554093`

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system, 2003.

[20] O. Görlitz and S. Staab. Splendid: SPARQL endpoint federation exploiting void descriptions. *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*. CEUR-WS. org, pages 13–24, 2011.

[21] O. Görlitz, M. Thimm, and S. Staab. Splodge: systematic generation of SPARQL benchmark queries for linked open data. *International Semantic Web Conference*. Springer, pages 116–132, 2012.

[22] D. Graux, L. Jachiet, P. Genevès, and N. Layaïda. A multi-criteria experimental ranking of distributed SPARQL evaluators, 2016.

[23] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[24] S. Harris, N. Lamb, and N. Shadbolt. 4store: the design and implementation of a clustered RDF store. *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pages 94–109, 2009.

[25] A. Harth, J. Umbrich, A. Hogan, and S. Decker. Yars2: a federated repository for querying graph structured data from the web. In: *The Semantic Web*, pages 211–224. Springer, 2007.

[26] D. Hernández, A. Hogan, C. Riveros, C. Rojas, and E. Zerega. Querying wikidata: comparing SPARQL, relational and graph databases. *International Semantic Web Conference*. Springer, pages 88–103, 2016.

[27] A. Jena. A free and open source java framework for building semantic web and linked data applications. *Available online: jena. apache. org/(accessed on 28 April 2015)*, 2015.

[28] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, and P. Castagna. Jena-hbase: a distributed, scalable and efficient RDF triple store. *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*. CEUR-WS. org, pages 85–88, 2012.

[29] A. R. Kinjo, H. Suzuki, R. Yamashita, Y. Ikegawa, T. Kudou, R. Igarashi, Y. Kengaku, H. Cho, D. M. Standley, A. Nakagawa, et al. Protein data bank japan (pdbj): maintaining a structural data archive and resource description framework format. *Nucleic acids research*, 2011.

[30] V. Kotsev, N. Minadakis, V. Papakonstantinou, O. Erling, I. Fundulaki, and A. Kiryakov. Benchmarking RDF query engines: the LDBC semantic publishing benchmark.

[31] G. Ladwig and A. Harth. CumulusRDF: linked data management on nested key-value stores. *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*, pages 30, 2011.

[32] A. Loizou, R. Angles, and P. Groth. On the formulation of performant {sparql} queries. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31:1–26, 2015.

[33] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL benchmark–performance assessment with real queries on real data. *The Semantic Web–ISWC 2011*, 2011.

[34] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of RDF data. *The VLDB Journal - The International Journal on Very Large Data Bases*, 19(1):91–113, 2010.

[35] A.-C. N. Ngomo and M. Röder. HOBBIT: holistic benchmarking for big linked data.

[36] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for RDF data. In. *The Semantic Web - ISWC 2006: 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings*. Ed. by I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. M. Aroyo. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pages 559–572.

[37] N. Papailiou, I. Konstantinou, D. Tsoumakos, P. Karras, and N. Koziris. H 2 rdf+: high-performance distributed joins over large-scale RDF graphs. *Big Data, 2013 IEEE International Conference on*. IEEE, pages 255–263, 2013.

[38] F. Picalausa and S. Vansummeren. What are real SPARQL queries like? *Proceedings of the International Workshop on Semantic Web Information Management - SWIM '11*, 2011.

[39] K. Rohloff and R. E. Schantz. High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store. *Programming Support Innovations for Emerging Distributed Applications*. ACM, pages 4, 2010.

[40] M. Saleem, A. Hasnain, and A.-C. N. Ngomo. Bigrdfbench: a billion triples benchmark for SPARQL endpoint federation.

[41] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. Ngonga Ngomo. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web*, 7(5):493–518, 2016.

[42] M. Saleem, Q. Mehmood, and A.-C. N. Ngomo. Feasible: a feature-based SPARQL benchmark generation framework. *International Semantic Web Conference*. Springer, pages 52–69, 2015.

[43] M. Saleem and A.-C. N. Ngomo. Hibiscus: hypergraph-based source selection for SPARQL endpoint federation. *European Semantic Web Conference*. Springer, pages 176–191, 2014.

[44] A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen. PigSPARQL: mapping SPARQL to pig latin. *Proceedings of the International Workshop on Semantic Web Information Management*. ACM, pages 4, 2011.

[45] A. Schätzle, M. Przyjaciel-Zablocki, A. Neu, and G. Lausen. Sempala: interactive SPARQL query processing on hadoop. *International Semantic Web Conference*. Springer, pages 164–179, 2014.

[46] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen. S2RDF: RDF querying with SPARQL on spark. *Proc. VLDB Endow.* 9(10):804–815, June 2016.

[47] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7031 LNCS(PART 1):585–600, 2011.

[48] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SPˆ 2Bench: a SPARQL performance benchmark. *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, pages 222–233, 2009.

[49] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. *The Semantic Web - ISWC 2011, Proceedings, Part I*, pages 601–616, 2011.

[50] F. Stegmaier, U. Gröbner, M. Döller, H. Kosch, and G. Baese. Evaluation of current RDF database solutions. *Proceedings of the 10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe), 4th International Conference on Semantics And Digital Media Technologies (SAMT)*. Citeseer, pages 39–55, 2009.

[51] Y. Tateno, T. Imanishi, S. Miyazaki, K. Fukami-Kobayashi, N. Saitou, H. Sugawara, and T. Gojobori. Dna data bank of japan (ddbj) for genome scale research in life science. *Nucleic acids research*, 30(1):27–30, 2002.

[52] K. Vandepoele. Dissecting plant genomes with the plaza 2.5 comparative genomics platform. 2013. `https://www.slideshare.net/klaasvandepoele/dissecting-plant-genomes-with-the-plaza-25-comparative-genomics-platform`

[53] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the Web with high availability. *Proceedings of the 13th International Semantic Web Conference*. Volume 8796 of Lecture Notes in Computer Science, pages 180–196, Springer, October 2014.

[54] X. Wang, G. Haberer, and K. F. Mayer. Discovery of cis-elements between sorghum and rice using co-expression and evolutionary conservation. *BMC genomics*, 10(1):284, 2009.

[55] H. Wu, T. Fujiwara, Y. Yamamoto, J. Bolleman, and A. Yamaguchi. BioBenchmark Toyama 2012: an evaluation of the performance of triple stores on biological data. *Journal of biomedical semantics*, 5(1):1, 2014.

[56] Y. Yamamoto, A. Yamaguchi, H. Bono, and T. Takagi. Allie: a database and a search service of abbreviations and long forms. *Database*, 2011, 2011.

[57] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A distributed graph engine for web scale RDF data. *Proc. VLDB Endow.* 6(4):265–276, February 2013.

[58] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. Gstore: answering SPARQL queries via subgraph matching. *Proceedings of the VLDB Endowment*, 4(8):482–493, 2011.

Part IV

Conclusions and Future Work

Chapter 8

Conclusion

> *It is possible to commit no mistakes and still lose. That is not a weakness, that is life.*
>
> —Jean-Luc Picard.

Replacing 'life' by 'PhD' in the above quote makes it quite relevant for the majority of doctoral dissertations: only a minor fraction results in a major breakthrough while most dissertations will not withstand the test of time. It is however necessary to try and approach a single problem from many different angles, even when we know that most of them will fail to deliver.

In my personal opinion the most substantial advances are the result of a collaboration between sets of motivated researchers with diverse backgrounds and a sufficient amount of time for ideas to mature . The i-ADHoRe algorithm is a good example of this. The maturity of the algorithm becomes clear if we look at the interval between the first ADHoRe [2] paper, published in 2002, until i-ADHoRe 3.0, which was published in 2011. The latest version also highlights the added value of having a diverse team. A good blend of statistics, insights in evolutionary biology and a focus on algorithmic optimization, and parallel computing, were the recipe for a significant step forward (**Hypothesis 1**). In terms of performance i-ADHoRe can easily process the largest datasets in less than an hour. The algorithm's superior sensitivity was shown in its ability to detect high-level Multiplicons and to reproduce well-known experimental results such as the detection of the HOX-cluster (**RQ1**).

On a pure technical level, the BLSSpeller algorithm, was certainly breakthrough. It was the first algorithm to tackle comparative motif discovery with an exhaustive

approach (**RQ2**) using a scalable distributed algorithm. This was again achieved on the ground of Hypothesis 1. While one could argue, that *we committed no mistakes*, the impact was less spectacular compared to the i-ADHoRe paper. After reading this dissertation this should however not come as a surprise: there is a serious barrier to continue this work, as the data was not published at all, only the method to generate the data. Back then, we did however lack the insights to do so properly. Today the most reasonable way to remedy this would be to resort to the FAIR data principles described in chapter 3.

In chapter 3 we introduced the concept of the Semantic Web to enable a global distributed database. Having self-descriptive data and automatic integration stimulates communication and transfer of ideas between diverse bodies of research, such as within the Life Sciences domain. The ideas of the Semantic Web on data publishing provide the exact solution for making the BLSSpeller motif database interesting to the research community, by providing an easy methodology to query the data, but for the publishers also a cheap way to publish the data, for example using the HDT-format [1].

There are however some technical hurdles to achieve a fluent interaction with this global information space. In chapter 7 we analyzed the ability of the current state-of-the-art in Semantic Web technologies for publishing and consuming Life Sciences data. In that context we developed a methodology to make the benchmark itself more reproducible (**RQ3**), by relying on reusable benchmark components, by using publicly available hardware, and by using docker and machine images for the tested systems. Furthermore the benchmark results are published in a sufficiently detailed format to allow additional research. For the non-disclosed benchmark we came up with a non-biased approach based on query features and decision trees to make the extract generalizable results. Relying on benchmark cost also enabled the comparison of completely different setups for publishing RDF (**RQ4**).

As a final word of advice we would like to stress the importance of data publication as part of the data science process (**Hypothesis 2**):

The results of applied pattern mining will make a much bigger impact if data publication becomes a primary concern in the research process and is not just treated as an afterthought.

Chapter 9

Future work

I'm burning through the sky. Two hundred degrees, that's why they call me Mister Fahrenheit. [...] Don't stop me now, I'm having such a good time...

—Farrokh "Freddie" Mercury

As was already recognized by Freddie Mercury, it is always hard to call it a day. The ideas in this chapter can serve as inspiration for future work and are, as is the case for this entire thesis, trying to improve on one of the components in the Big Data interactions Figure 2.2 on page 11.

9.1 Improvements to the BLSSpeller algorithm

Alternative Indexing Scheme

BLSSpeller uses one GST per gene family. This choice was originally made, to minimize the amount of memory required to store the GST, to leave more space available for storing the motifs.

As BLSSpeller deals with one gene family at a time (per mapper), only one index structure must fit in memory: when one gene family is processed the associated GST can be removed.

This has no negative impact on the memory and time complexity for the GST, as these scale with the total sequence length: $\mathcal{O}(S_{tot})$. When the algorithm was rewrit-

ten for the MapReduce framework, motifs suddenly could be stored on disk. This creates new avenues for optimizing the BLSSpeller algorithm.

In the master thesis of Gilles Jacobs [11], the feasibility of an alternative indexing scheme was analyzed: Build a GST for all gene families at once, in every mapper. This new GST, which we will call *BigTrie*, must only be partially built for every mapper. The associated parallelization scheme now partitions the computations in the IUPAC pattern space: Each mapper only emits motifs with a certain prefix.

An important drawback of this approach is, that the bitvectors stored in the internal nodes of the GST, now have to represent both a gene family id and a sequence number. This additional overhead is even stronger, as the GST will be more densely populated, i.e. it will have internal nodes at larger string depths as compared to the single gene family approach. As a rule of thumb we consider that the GST is dense up until a string depth $k_{dense} = \arg\max_k (4^k \leq S_{tot})$. For S_{tot} this means $k_{dense} \approx 7$, for BigTrie this will be $k_{dense} \approx 14$. The main advantage of this approach is however that in a single lookup the entire frequency vector $F(T)$ can be calculated. In the current implementation of BLSSpeller $F(T)$ is only aggregated in the reducer after shuffling all partial results over the network.

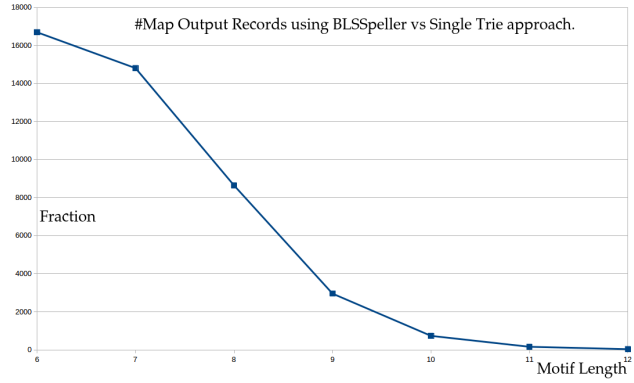


Figure 9.1: Overhead is calculated as the total amount of candidate motifs emitted by BLSSpeller in the map phase versus the maximal number of unique motifs in the IUPAC search space. The latter is the upper boundary in the single trie approach. For $k = 12$, still a compression of a factor 33 is obtained. Aggregated over all motif lengths the amount of data sent over the network could be lowered by at least a factor 97.

Let's make an estimation on the amount of network overhead that can be saved by using the BigTrie setup. The output of the speller algorithm is dominated by

$k = 12$ motifs as became apparent in Figure 6.16. These motifs often occur only in a single gene family. For these motifs this new setup would therefore have no advantage. However, it turns out that actual overhead for motifs with $k_{max} = 12$, although converging to 0, is far from negligible as can be seen in Figure 9.1. All gains added together, the BigTrie implementation can lower the network overhead by two orders of magnitude.

It might even be feasible to take this one more step further. The entire motif discovery algorithm could in theory also be run without any network communication. That would mean that every mapper searches for all motifs which occur in a single permutation group. The feasibility of this approach however, requires additional research.

Conservation Metrics

One limitation of the Confidence score $C(T)$ that became apparent when studying the motifs that satisfy the $C(T) > C_{thresh}$ constraint, is that only few short motifs satisfy this criterion.

The short and/or degenerate motifs tend to be part of permutation groups for which F_{bg} is high. But $F(T)$ is bounded by the amount of gene families G: $\max F(T) = G$. Therefore the highest F_{bg} for which we can find patterns:

$$\frac{G}{F_{bg}^{max}} = 1 - C_{thresh}$$

For $C_{thresh} = 0.9$ this correspond to $F_{bg}^{max} = 1772$ There are however experimentally validated binding sites with far more putative binding sites in the Monocot dataset.

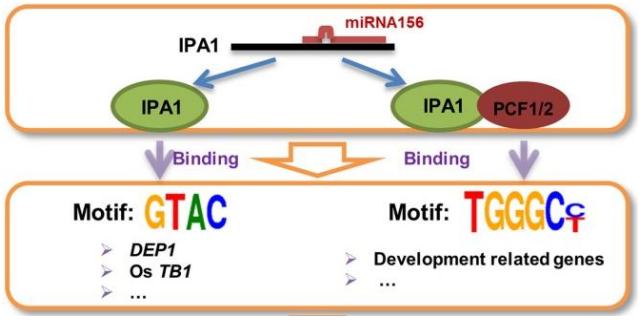


Figure 9.2: The IPA1 transcription factor binds to directly to the GTAC and indirectly to TGGGCC/T binding sites.

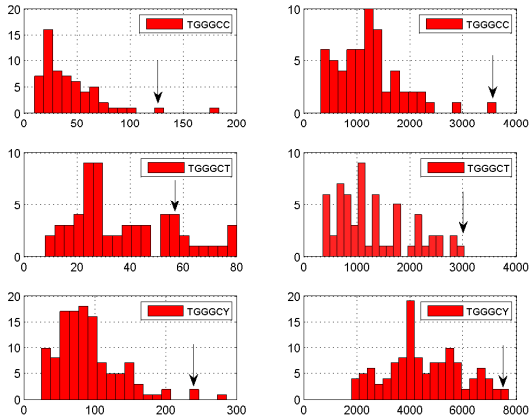


Figure 9.3: Experimental distribution of F_{bg} for the permutation groups associated with the 3 IUPAC motifs which match with the IPA1 binding site. On the left hand site the distributions for AB discovery are shown, on the right hand site for AF motif discovery. Although the binding site always lie in the outlier of the distributions their family occurrence F is usually only 2 times that of the median value and therefore $C_{thres} \approx 0.5$

An example of a biologically known short binding site is that of the IPA1 transcription factor in Rice. This transcription factor plays an important role in the plant architecture of Rice (Oryza sativa) , as was discovered by Lu [13]. The factor binds to a GTAC motif, but this binding is further affected by indirect bindings to TGGGCC/T as shown in Figure 9.2.

In Figure 9.3 we analyze the distribution of F_{bg} for all permutations of the 3 corresponding IUPAC strings. Although the IPA1 binding sites are in the deep tail of the distribution of $F_{bg}(T)$, they would not surpass the C-threshold.

One method could be extend the BLSSpeller algorithm to look for *structured motifs*, i.e a combination of motifs, for which we can again be inspired by follow-up work by Sagot [14]. Alternatively we can abandon the usage of the confidence thresholds altogether and opt for a trivial approach, which only reports the motifs in the outliers of the distribution: the Top-N motifs per permutation group could for example be retained.

Apart from the Confidence scores also the BLS metric has its limitations as was already proven by Xie [17]. In their research paper, they suggested the use of a

Bayesian Branch Length Score, which takes into account that some occurrences in a gene family might still be attributed to chance. Another limitation of the BLS score is that different gene combinations can be mapped to the same BLS score. We have already shown [9], that for the case of only a limited number of species it might therefore be beneficial, in terms of the algorithm's precision, to opt for a species combination score (SC). This is simply the integer representation of the bit vector associated with a motif.

From MapReduce to Spark?

The BLSSpeller algorithm was parallelized using version 1 of the Hadoop framework. While the source code is compatible with newer versions of the Hadoop framework, a new version of the BLSSpeller algorithm should definitely be implemented using Apache Spark.

The reason is, that this framework beats the original MapReduce approach on the Terasort benchmarks[1].

Even more important is the usability of the Spark framework. Spark supports a much wider range of workloads than the original MapReduce: batch processing, machine learning, interactive querying and stream processing. Spark also supports more common data science language such as Python (PySpark) which enables faster prototyping of parallel data analysis tasks such as BLSSpeller. Since the introduction of the Dataframe API, the choice of programming language no longer has a negative impact on the performance [2].

The usage of this API would minimize the amount of custom code for BLSSpeller: only the mapper should be implemented as a user-defined function. The remainder of BLSSpeller could be re-written mainly relying on the GROUP BY operator in the Dataframe API.

9.2 Pattern Mining in the BLSSpeller Motif Database

Clustering of Redundant Patterns and Discovery of Structural Motifs

The amount of motifs reported by BLSSpeller is such, that it should be called a *motif database*. This rich output creates opportunities for further analysis.

[1] `http://sortbenchmark.org/`

In the analysis of the KN1 transcription factor, it already became clear that many IUPAC variants of the KN1 sequence logo are nonessential. This means, that for each of the gene families for which it has a match with experimentally verified binding sites of Bolduc, there is another IUPAC motif which has higher values for $F(T_i)$ and/or $C(T_i)$. In Figure 9.4 we ran a test using a naive greedy clustering algorithm which eliminated all nonessential motifs from as set of 432 IUPAC variants which match with a set of KN1 target genes. Of this set of variants only 14 variants are essential. This is a strong indication that there must be a way to cluster these variants together.

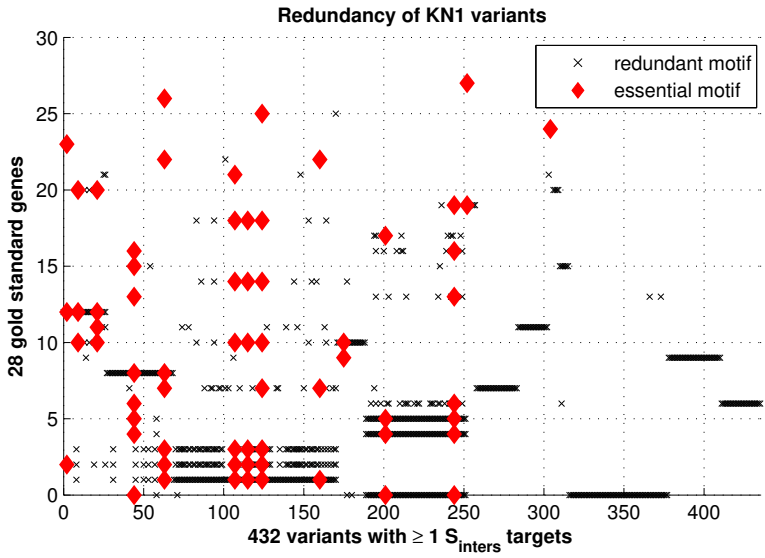


Figure 9.4: Graphical representation of greedy clustering approach to select 14 essential motifs to cover a set of 28 KN1 binding sites (identified by Bolduc). Each cross indicates the match between a nonessential KN1 variant and one of genes. The red diamonds indicate matches whereby the variant is considered essential.

One approach that seems novel, is to use certain standard pattern mining algorithms on a large matrix, in which every IUPAC motif is represented by a row and for which every column represents a target gene family. In this representation classical dimensionality reduction techniques (Principal Component Analysis for example) can be used to find genes which are mainly targeting the same gene families.

Also interesting, is the fact that the results of clustering in the row space of this matrix could lead to clusters for which motifs are not necessarily within a small mu-

tual edit distance. This clustering would thus enable the identification of structured motifs (multiple binding sites).

The clustering could also be performed in the column space to detect gene families which form *modules*. Standard computational approaches for the discovery of genetic regulatory modules usually rely on gene expression data [3]. Finally the clustering algorithm can also be run in both spaces simultaneously. This corresponds to a type of algorithms called *bi-clustering*, one example being the Pro Bic algorithm [18] developed by IDLab colleagues.

Deep Learning for Motif Discovery

The de-novo discovery has many similarities with gene target prediction (given a motif). Also the BLS approach was initially used for gene target prediction, prior to its adoption for de-novo motif discovery.

Probabilistic approaches to motif discovery have long been restricted to variations on Gibbs Sampling and Expectation Maximization. The spectacular breakthroughs with deep neural networks, also for sequence models [12], also led to some attempts at studying there applicability for ohmic sequences.

In this context the work on the DeepBind [1] can serve as a starting point, for designing an algorithm for comparative motif discovery. It would be interesting to verify whether training a deep neural network, with the BLSSpeller output as the training set, would for example be more successful at predicting OCR-regions.

9.3 Linked Data Genome Browser

The collaboration with Ontoforce demonstrated the added value of data integration for the Life Sciences domain. An important insight, from assessing the validity of the BLSSpeller algorithm, is that this assessment always come from cross referencing with results from different experiments.

For example, a standard practice to study the validity of motif discovery results is GO Enrichment Analysis [2]. Given a database with functional annotations of genes (GO terms), one can analyze whether the target genes of a certain putative motif, have overrepresented GO terms. This increases the chance that the putative motif is in fact a biological signal.

[2] `http://geneontology.org/page/go-enrichment-analysis`

The validation of motif discovery results could therefore be automated if we *publish the motif database as Linked Data*. In the past years IDLab has made significant of progress in simplifying the conversion of any type of data to RDF, see for example the efforts of Dimou [10]. Some widely adopted file formats in bioinformatics for which RML descriptions would have to be generated are formats for sequence annotations such as BED, GFF3 and VCF and the sequences themselves such as FASTA (nucleotide and peptide sequences), SAM (aligned sequences), …

This mapping process, does require a target vocabulary. The Gene Ontology project [7] provides a set of OWL-based vocabularies for annotating ohmic sequences as well as database where these annotations can be published [6]. A number of ontologies which can be explored as a starting point are the Sequence Ontology [3]), which can for example annotate a sequence as being a `binding_site` and the FALDO [5] ontology for describing locations in ohmic sequences.

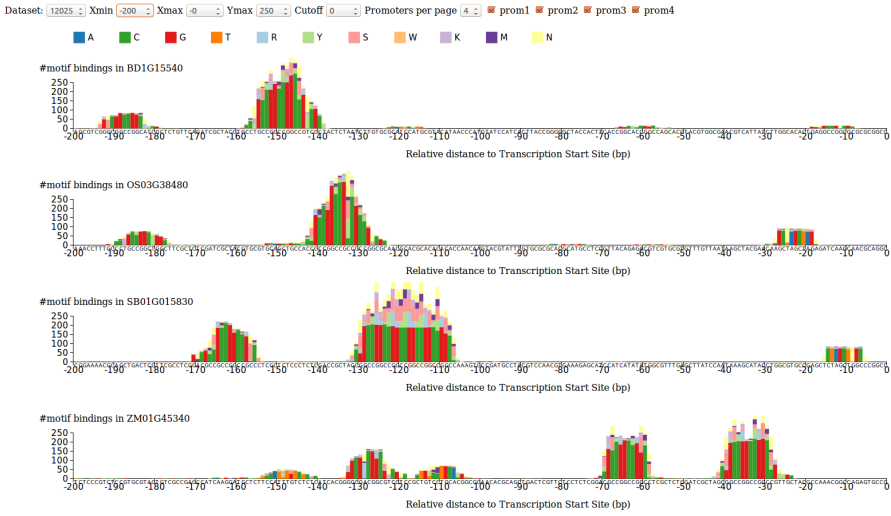


Figure 9.5: An initial attempt at creating a front-end for a genome browser interface for the motif database generated by the BLSSpeller algorithm. The interface has handles to explore the different gene families, and to select a window of interest in the promoter sequences. The colors give insight into the composition of the motifs which have a target in the promoter sequences. In this example we see a number of highly targeted regions correspond to 'CCGG' repeats.

One of the largest RDF datasets in the Semantic Web is the UniProt database [8]. As the goal of gene transcription is often to generate proteins, an interesting future

[3] `http://www.sequenceontology.org/`

research direction could be study the added value of integrating this data and to use it in a motif discovery workflow.

Once the data is available as RDF, we can use our insights gathered in chapter 7 on how to publish Linked Data and how to run federated queries against it. In this context it would be interesting to explore what Linked Data Fragments [16] are most suitable to interact with genomic data. Some initial work has been done by Taelman [15] where LDF interfaces for ordinal data (such as genomic sequences) were studied.

Requirements for publishing the motif database The main requirements of such a system for data publication would be the support for a FILTER operator. This operator would allow us to filter the motif database on a different set of parameters such as the motif length, degeneracy, C, T and F. With sufficient preprocessing the HDT-format might suffice to accomplish this goal. In terms of queries related to genome browsing it might however be interesting to use a RDF database solution with support for full-text search and for selecting genomic sequence 'intervals'. This functionality can definitely be supported if the RDF system supports spatial search operators. For a review of RDF systems in terms of full-text and spatial support we refer to Bellini [4].

The penultimate goal of this data integration is to provide a exploratory user interface which translates user actions in the front-end to federated queries on the Life Sciences cloud.

We already initiated this work by creating a minimalistic genome browser[4] which shows where the BLSSpeller motifs, which satisfy a certain combination of filter criteria (F_{thresh} and C_{thresh}), are located in the gene families. A screenshot of this initial attempt is given in Figure 9.5. This visualization tries to provide more insights about where the motifs accumulate as their positions correlation with OCR regions.

Additional considerations for general Life Sciences data publication When studying the data related to plants or animals, privacy might not be a huge concern. However, as pointed out in the introduction of this book, personalized medicine might be one of the largest sources of Big Life Sciences Data in the near future. The recently introduced General Data Protection Regulation (GDPR), imposed by the

[4] `https://github.com/drdwitte/CloudSpellerFrontend`

European Union as of May 2018, creates some safety mechanisms as to what can be done with privacy sensitive data. This law has led to a strong reaction by the Data Science community, which is afraid it will make it more difficult to innovate in this area as opposed to other countries where the GDPR regulation does not apply.

I would however like to conclude on a more positive note by demonstrating that in fact this regulation might result in more data innovation in the long run and might give rise to a set of new interesting research questions. Tim Berners-Lee in fact has been advocating the use of *personal data pods*, where all the data which is about an end-user is in fact owned by this user. This requires the current data applications to be re-engineered, such that the data, which is now centralized, is decoupled from the data applications. This decentralization would tackle most of the challenges imposed by the GDPR law. As data would no longer be locked away in data silos, such as Facebook and Google, this could even lead to more innovation and competition in the market for data applications, as newcomers no longer face the 'cold-start' issue of having no access to user data. An interesting link here is to point to the Solid project[5] which is about the personal data management. A visionary blog post of Ruben Verborgh, titled paradigm shifts for the decentralized Web[6], demonstrates how a lot of already available components in the Semantic Web stack can be brought together to accomplish this challenge.

Also in the context of Life Sciences data these ideas are beginning to crystallize. An interesting project by the FAIR data movement is the Personal Health Train (PHT) project[7]. There data is stored in FAIR data stations, each with their own access license. Trains in this metaphor correspond to data science workflows, which try to answer certain research questions. From the viewpoint of the data scientist it might be very interesting to map the classical data science workflow on this new paradigm. This would correspond to even more decentralization by moving to decentralized data mining and machine learning, which will without any doubt be the basis for many interesting future research questions.

[5] The Solid project: `https://solid.mit.edu/`
[6] Blog: `https://ruben.verborgh.org/blog/2017/12/20/paradigm-shifts-for-the-decentralized-web/`
[7] PHT: `https://www.dtls.nl/fair-data/personal-health-train/`

References

[1] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831, 2015.

[2] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: relational data processing in spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, pages 1383–1394, 2015.

[3] Z. Bar-Joseph, G. K. Gerber, T. I. Lee, N. J. Rinaldi, J. Y. Yoo, F. Robert, D. B. Gordon, E. Fraenkel, T. S. Jaakkola, R. A. Young, et al. Computational discovery of gene modules and regulatory networks. *Nature biotechnology*, 21(11):1337, 2003.

[4] P. Bellini and P. Nesi. Performance assessment of rdf graph databases for smart city services. *Journal of Visual Languages & Computing*, 45:24–38, 2018.

[5] J. T. Bolleman, C. J. Mungall, F. Strozzi, J. Baran, M. Dumontier, R. J. Bonnal, R. Buels, R. Hoehndorf, T. Fujisawa, T. Katayama, et al. Faldo: a semantic standard for describing the location of nucleotide and protein feature annotation. *Journal of biomedical semantics*, 7(1):39, 2016.

[6] G. O. Consortium. The gene ontology (go) database and informatics resource. *Nucleic acids research*, 32(suppl_1):D258–D261, 2004.

[7] G. O. Consortium. The gene ontology project in 2008. *Nucleic acids research*, 36(suppl_1):D440–D444, 2007.

[8] U. Consortium. Uniprot: a hub for protein information. *Nucleic acids research*, 43(D1):D204–D212, 2014.

[9] D. De Witte, M. Van Bel, P. Demeester, B. Dhoedt, K. Vandepoele, and J. Fostier. Alignment-free genome-wide comparative motif discovery in 4 monocot species. eng. *11th European Conference on Computational Biology, Abstracts*, pages 1–1, Basel, Switzerland, 2012.

[10] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. Rml: a generic language for integrated rdf mappings of heterogeneous data. *LDOW*, 2014.

[11] G. Jacobs. Gedistribueerde motiefdetectie met mapreduce. MA thesis. Ghent University, 2014.

[12] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[13] Z. Lu, H. Yu, G. Xiong, J. Wang, Y. Jiao, G. Liu, Y. Jing, X. Meng, X. Hu, Q. Qian, et al. Genome-wide binding analysis of the transcription activator ideal plant architecture1 reveals a complex network regulating rice plant architecture. *The Plant Cell*, 25(10):3743–3759, 2013.

[14] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of computational biology*, 7(3-4):345–362, 2000.

[15] R. Taelman, P. Colpaert, R. Verborgh, and E. Mannens. Multidimensional interfaces for selecting data within ordinal ranges. *Proceedings of the 7th International Workshop on Consuming Linked Data co-located with 15th International Semantic Web Conference (ISWC 2015)*, 2016.

[16] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the Web with high availability. *Proceedings of the 13th International Semantic Web Conference*. Volume 8796 of Lecture Notes in Computer Science, pages 180–196, Springer, October 2014.

[17] X. Xie, P. Rigor, and P. Baldi. Motifmap: a human genome-wide map of candidate regulatory motif sites. *Bioinformatics*, 25(2):167–174, 2008.

[18] H. Zhao, L. Cloots, T. Van den Bulcke, Y. Wu, R. De Smet, V. Storms, P. Meysman, K. Engelen, and K. Marchal. Query-based biclustering of gene expression data using probabilistic relational models. *BMC bioinformatics*, 12(1):S37, 2011.

Appendices

Appendix A

Publications

A.1 Journal Publications - A1

- **Reproducible Query Performance Assessment of Scalable RDF Storage Solutions**
  Dieter De Witte, Laurens De Vocht, Dieter De Paepe, Jan Fostier, Filip Pattyn, Kenny Knecht, Hans Constandt, Ruben Verborgh and Erik Mannens (2018) Journal of Biomedical Semantics. Submitted

- **BLSSpeller: exhaustive comparative discovery of conserved cis-regulatory elements.**
  Dieter De Witte, Jan Van de Velde, Dries Decap, Michiel Van Bel, Pieter Audenaert, Piet Demeester, Bart Dhoedt, Klaas Vandepoele and Jan Fostier (2015) BIOINFORMATICS. 31(23). p.3758-3766

- **i-ADHoRe 3.0 : fast and sensitive detection of genomic homology in extremely large data sets.**
  Sebastian Proost, Jan Fostier, Dieter De Witte, Bart Dhoedt, Piet Demeester, Yves Van de Peer and Klaas Vandepoele (2012) NUCLEIC ACIDS RESEARCH. 40(2).

A.2 Conference Publications - P1

- **A parallel, distributed-memory framework for comparative motif discovery.**
  Dieter De Witte, *Michiel Van Bel, Pieter Audenaert, Piet Demeester, Bart Dhoedt, Klaas Vandepoele and Jan Fostier* (2014) Lecture Notes in Computer Science. 8385. p.268-277

- **Simulation of RF-fields in a fusion device** Dieter De Witte, *Ignace Bogaert, Daniël De Zutter, Guido Van Oost and Dirk Van Eester* (2009) AIP Conference Proceedings. In AIP Conference Proceedings 1187. p.593-596

A.3 Conference Publications - C1

- **Scaling out federated queries for life sciences data in production.**
  Dieter De Witte, *Laurens De Vocht, Filip Pattyn, Hans Constandt, Erik Mannens and Ruben Verborgh* (2016) SWAT4LS. p.1-10

- **Big linked data ETL benchmark on cloud commodity hardware.**
  Dieter De Witte, *Laurens De Vocht, Ruben Verborgh, Kenny Knecht, Filip Pattyn, Hans Constandt, Erik Mannens and Rik Van de Walle* (2016) Proceedings of the International Workshop on Semantic Big Data.